SANDxxxx-xxxx
Unlimited Release
Printed May 2009

# OVIS 2.0.3 User's Guide

J. Brandt
Sandia National Laboratories
M.S. 9152, P.O. Box 969
Livermore, CA 94551, U.S.A.
brandt@sandia.gov

A. Gentile
Sandia National Laboratories
M.S. 9152, P.O. Box 969
Livermore, CA 94551, U.S.A.
gentile@sandia.gov

J. Mayo
Sandia National Laboratories
M.S. 9159, P.O. Box 969
Livermore, CA 94551, U.S.A.
jmayo@sandia.gov

P. Pébay
Sandia National Laboratories
M.S. 9159, P.O. Box 969
Livermore, CA 94551, U.S.A.
pppebay@sandia.gov

D. Roe
Sandia National Laboratories
M.S. 9152, P.O. Box 969
Livermore, CA 94551, U.S.A.
dcroe@sandia.gov

D. Thompson
Sandia National Laboratories
M.S. 9152, P.O. Box 969
Livermore, CA 94551, U.S.A.
dcthomp@sandia.gov

M. Wong
Sandia National Laboratories
M.S. 9152, P.O. Box 969
Livermore, CA 94551, U.S.A.
mhwong@sandia.gov

**Abstract**

This document[1] describes how to obtain, install, use, and enjoy a better life with OVIS version 2.0.

[1] Last Revision:July 13, 2009

# Contents

# Figures

# 1 Introduction

This document is the user's guide for OVIS version 2.0.

The OVIS project [5] targets scalable, real-time analysis of very large data sets. We characterize the behaviors of elements and aggregations of elements (e.g., across space and time) in data sets in order to detect anomalous behaviors. We are particularly interested in determining anomalous behaviors that can be used as advance indicators of significant events of which notification can be made or upon which action can be taken or invoked.

The OVIS open source tool (BSD license) is available for download at ovis.ca.sandia.gov. While we intend for it to support a variety of application domains, the OVIS tool was initially developed for, and continues to be primarily tuned for, the investigation of High Performance Compute (HPC) cluster system health. In this application it is intended to be both a system administrator tool for monitoring and a system engineer tool for exploring the system state in depth.

OVIS 2.0 provides a variety of statistical tools for examining the behavior of elements in a cluster (e.g., nodes, racks) and associated resources (e.g., storage appliances and network switches). It calculates and reports model values and outliers relative to those models. Additionally, it provides an interactive 3D physical view in which the cluster elements can be colored by raw element values (e.g., temperatures, memory errors) or by the comparison of those values to a given model. The analysis tools and the visual display allow the user to easily determine abnormal or outlier behaviors.

The OVIS project envisions the OVIS tool, when applied to compute cluster monitoring, to be used in conjunction with the scheduler or resource manager in order to enable intelligent resource utilization [2, 3]. For example, nodes that are deemed less healthy, that is, nodes that exhibit outlier behavior in some variable, or set of variables, that has shown to be correlated with future failure, can be discovered and assigned to shorter duration or less important jobs. Further, applications with fault-tolerant capabilities can invoke those mechanisms on demand, based upon notification of a node exhibiting impending failure conditions, rather than performing such mechanisms (e.g. checkpointing) at regular intervals unnecessarily.

More information about the OVIS project and publications describing the OVIS research in more detail can be found at ovis.ca.sandia.gov [5].

The OVIS team can be reached at ovis-help@sandia.gov.

This page left intentionally blank

# 2 Installation

This section contains build instructions for the OVIS release. You will be required to obtain and install supporting software that is not part of OVIS in a manner appropriate to your platform. Necessary supporting software is listed below. System settings for running OVIS are also given.

## 2.1 Supporting Software

1. Obtain and install a C compiler, a C++ compiler, OpenGL version 1.2 or newer. These are probably already installed on your system.
2. Obtain and install perl-DBI
3. Obtain and install a database - either MySQL or PostgreSQL will work (As of this writing the current rpms for Fedora will work):

   MySQL:

   (a) Obtain and install MySQL at least version 5.0.51 (install client, server, devel, and shared)
   (b) Obtain and install MySQL-python

   PostgreSQL:

   (a) Obtain and install PostgreSQL at least version 8.2.9
   (b) Set the following in your environment:

   ```
   setenv PGSQL_HOME /usr/local/pgsql
   set path=($PGSQL_HOME/bin \$path)
   ```

4. Obtain and install Qt using the latest stable version of 4.3.x. Configure with

   ```
   -plugin-sql-mysql -plugin-sql-psql -debug
   ```

   As of this writing the current rpm for Fedora will work (Qt version 4.3.x). You will need the qt-devel rpm and the rpm for the appropriate database plugin as well.
5. Obtain and install CMake at least version 2.6.0
6. Obtain, build and install VTK:

   (a) Obtain VTK [4]. You may use either version 5.4 or the latest CVS trunk version of the VTK source via CVS. See the instructions to "Access the CVS source-code repository" at http://www.vtk.org/get-software.php.
   (b) If you installed Qt from source, you may need to tell ccmake where to find Qt:

   ```
   setenv QTDIR /path/to/Qt-4.3.3
   ```

(The default location is `/usr/local/Trolltech/Qt-4.3.3`)

(c) Build and install VTK:

```
mkdir /path/to/vtkBuildDir
cd /path/to/vtkBuildDir
ccmake /path/to/vtkSourceDir
```

(d) Use the following settings within ccmake:
(Type 'c' to configure to see these options)

BUILD_EXAMPLES: ON (This setting is optional.)

BUILD_SHARED_LIBS: ON

CMAKE_BUILD_TYPE: Debug (Type in)

**The following options may require typing 't' for advanced mode to see, and may appear in an iterative fashion as items are selected:**

VTK_USE_MYSQL: ON

VTK_USE_POSTGRES: ON

CMAKE_CXX_FLAGS_DEBUG: -g -Wall -Wextra -W -Wshadow
-Wunused-variable -Wunused-parameter -Wunused-function -Wunused
-Wno-system-headers -Wno-deprecated -Woverloaded-virtual

VTK_USE_GUISUPPORT: ON

VTK_USE_QVTK: ON
(To access this option, press 'c' to configure after setting VTK_USE_GUISUPPORT)

DESIRED_QT_VERSION: 4
(To access this option, press 'c' to configure again after setting VTK_USE_QVTK)

(e) Now you should have a 'g' to generate option. If not, type 'c' to configure again.

(f) Type 'g' to generate.

(g) `make`

(h) `make install`

7. Obtain and install libdaemon.

8. Obtain and install dbus.

9. Obtain and install avahi at least version 0.6.22. This may require you to get intltool for the install. On 64-bit systems, you will need either version 0.6.23 or the patched version of Avahi included with OVIS. Whatever version of Avahi you use, you must build the avahi-qt4 library if you plan to build the OVIS baron GUI.

10. Build avahi and install it as a system service. Configure avahi as follows:

```
setenv PKG_CONFIG_PATH /usr/local/lib/pkgconfig:$QTDIR/lib/pkgconfig
./configure --disable-python-dbus --disable-mono --disable-python \
   --disable-gtk --disable-qt3
```

## 2.2 OVIS Install

1. Obtain the OVIS source at ovis.ca.sandia.gov
2. `mkdir /path/to/OVISBuildDir`
3. `cd /path/to/OVISBuildDir`
4. `ccmake /path/to/OVISSrcDir`
5. Use the following settings within ccmake:
   (Type 'c' to configure to see these options; ignore initial warnings)

   CMAKE_BUILD_TYPE: Debug

   DNS_SD_INCLUDE_DIRECTORIES: `/usr/include` (or the include dir of Avahi if you installed it in a non-standard location)

   OVIS_USE_AVAHI: ON

   VTK_DIR: `/path/to/VTK/BUILD/dir` (Don't use the VTK install dir)

   Type 'c' to configure. You may see more warnings but these may be ignored if there are no conflicts.

   Make sure that all the Avahi library variables point to the versions in `/usr/local/lib` and not `/usr/lib64` or `/usr/lib` libraries if you have built your own version of Avahi.
6. Now you should have a 'g' to generate option. If not, type 'c' to configure again.
7. Type 'g' to generate.
8. `make`
9. `make install`

You are now done building OVIS 2.0!

Note also that OVIS will place on your system the plain text configuration file `${HOME}/.config/Sandia/ovis.conf`. This will contain Baron state information (as described in §6) and database usernames and passwords should you choose to explicitly save them (see the ServerConnection window in Figure 14 described in §6 and §7.1).

## 2.3 MySQL Settings

1. If MySQL is not currently running (however, it should be if you are running on a system with Fedora 8 or greater installed):

```
/sbin/chkconfig --level 345 mysqld on # run MySQL daemon at boot
/sbin/service mysqld start # run MySQL daemon now
```

2. You should create a database user named ovis. This user will need full administrative privileges on the local machine and the ability to alter tables and insert records from machines where data will be collected. The administrative privileges are required to load a shared library (`libovis-mysql.so`) that contains functions used to signal the OVIS shepherd process when rows in certain tables are inserted or modified. For connections from network interfaces that face outside the cluster, you may require a password for the ovis user. Remote connections from external networks will still require permission to insert records into the database in order to request analyses of collected data. For example, if you will be using OVIS to store data into a database called OVIS_Cluster:

```
mysql -u root -p
mysql> GRANT ALL PRIVILEGES ON OVIS_Cluster.* TO 'ovis'@'localhost';
mysql> GRANT ALL PRIVILEGES ON OVIS_Cluster.* TO 'ovis'@'192.168.1.%';
mysql> GRANT INSERT,DELETE,UPDATE,EXECUTE,SELECT ON OVIS_Cluster.* TO
 'ovis'@'someremotehost' IDENTIFIED BY 'somepassword';
mysql> flush privileges;
```

   Note that you should configure mysqld so that no password is required to access the database from the private (administrative) network of your cluster or the local host where mysqld and the OVIS shepherd process will run. This way, sheep and shepherd processes do not need to be configured with any database passwords.

3. There is a user defined function that must be loaded into the MySql database. This will go into the mysql.func table. It needs only to be added once – not on a per OVIS_Cluster basis. If you use the database effector described in §4.2, this will occur with the `-t 4` flag. If you load a mysqldump (such as the example one for the Whitney database that comes with the release), ideally the function should be loaded as part of this process, however in practice the authors have seen that this may be dependent on the MySQL version. It is thus recommended that you run the database effector at this point.

4. Note: Some users have reported that VTK's MySQL interface cannot find `mysql.sock` despite its location being specified in the `/etc/my.cnf` configuration file. If this occurs, do the following:

```
cd /tmp
ln -s /var/lib/mysql/mysql.sock mysql.sock
```

## 2.4 PostgreSQL Settings

If you wish to use a PostgreSQL database to hold OVIS information, you will need to configure it appropriately.

1. You will need to edit the configuration file `/var/lib/pgsql/data/postgresql.conf` and change the `listen_addresses` setting to allow incoming connections from remote machines (both sheep inserting measurements of cluster behavior and users connecting with the baron to perform analysis). Unless you have a reason to specifically avoid a particular network interface, we suggest listening on all interfaces. We also recommend turning off informational messages printed by clients.

   ```
   listen_addresses = '*' # listen on all network interfaces
   client_min_messages = warning # print only warnings+errors
   ```

2. In addition to requesting that the daemon listen on network interfaces, you must specify how authentication should occur in `/var/lib/pgsql/data/pg_hba.conf` . For local connections or from network interfaces on the administrative network of a cluster, you should require no password so that sheep and shepherd processes may connect. For connections from network interfaces that face outside the cluster, you may require a password for the ovis user. As an example, consider the following lines:

   ```
   # TYPE  DATABASE     USER  CIDR-ADDRESS          METHOD
   ## Connections on the local machine
   local   ovis         ovis                        trust
   host    ovis         ovis  127.0.0.1/32          trust
   local   OVIS_Cluster ovis                        trust
   host    OVIS_Cluster ovis  127.0.0.1/32          trust
   ## Connections on private cluster admin network
   host    OVIS_Cluster ovis  192.168.1.254/24      trust
   ## Connections from remote sites. Requires password
   host    OVIS_Cluster ovis  74.125.19.19/24       ident
   ```

3. If the PostgreSQL postmaster daemon was running and you changed any of the configuration files above, you should restart it:

   ```
   /sbin/service postgresql restart
   ```

   Otherwise, if the daemon was not currently running, set it to run on reboot and then start it manually:

   ```
   /sbin/chkconfig --level 345 postgresql on # run daemon at boot
   /sbin/service postgresql start # run PostgreSQL daemon now
   ```

4. You should create a database user named ovis. This user will need full administrative privileges on the local machine and the ability to alter tables and insert records from machines where data will be collected. The administrative privileges are required to load a shared library (`libovis-psql.so`) that contains functions used to signal the OVIS shepherd process when rows in certain tables are inserted or modified. For example, if you will be using OVIS to store data into a database called OVIS_Cluster:

```
createuser -s -d -l -P ovis  # You will be prompted for a password.
createdb -p -U ovis ovis     # Enter the password for ovis.
createdb -p -U ovis OVIS_Cluster
```

5. There is a user defined function that must be loaded into the PostgreSQL database. It needs only to be added once – not on a per OVIS_Cluster basis. If you use the database effector described in §4.2, this will occur with the `-t 4` flag.

## 2.5 Additional General System Settings

Finally, many Linux distributions will need some system settings changed, links created, and daemons turned on or off.

1. While it is possible to run the shepherd process on systems with SELinux enabled, it is beyond the scope of this document to cover all of the configuration issues required. You may wish to configure your shepherd nodes to run in permissive rather than enforcing mode.

2. Place the following lines in your iptables configuration file (`/etc/sysconfig/iptables` on most systems):

   ```
   # Allow mDNS (also known as Avahi, Zeroconf, Bonjour)
   -A INPUT -m state --state NEW -m udp -p udp --dport 5353 \
       -d 224.0.0.251 -j ACCEPT
   # OVIS
   -A INPUT -m state --state NEW -m udp -p udp --dport 49154 -j ACCEPT
   -A INPUT -m state --state NEW -m tcp -p tcp --dport 53170 -j ACCEPT
   ```

   You may also need to add entries to allow PostgreSQL (port 5432) or MySQL (port 3306) connections, depending on which distribution of Linux you use and which database you prefer.

3. Turn libvirtd off

4. Set symbolic links for libraries if you are on a 64 bit machine:

   ```
   cd /usr/lib64
   ln -s /path/to/ovisBuildDir/lib/libovis-mysql.so libovis-mysql.so
   ln -s /path/to/vtkInstallDir/lib/vtk-5.3/libvtkCommon.so.5.3 \
     libvtkCommon.so.5.3
   ln -s /path/to/vtkInstallDir/lib/vtk-5.3/libvtkFiltering.so.5.3 \
     libvtkFiltering.so.5.3
   ln -s /path/to/vtkInstallDir/lib/vtk-5.3/libvtksys.so.5.3 \
     libvtksys.so.5.3
   ```

# 3 OVIS Components

In this section we describe the components of OVIS and their general interaction. For more information on the OVIS architecture see [1].

## 3.1 OVIS Components

OVIS uses a whimsical and intuitive analogy for naming its components.

- **Sheep** - the components which report (bleat) data values. Details on setting up the sheep data collectors can be found in §4.3.
- **Shepherds** - the components which maintain the databases to which the sheep report. Details on setting up the database can be found in §4.1 and §4.2.
- **Baron** - the GUI from which the data can be viewed and analyzed. Features of the Baron are given in §6 with an example in §7.1.
- **Haruspices (singular Haruspex)** - analysis engines, used to determine potentially portentous abnormal behaviors. These are named after *haruspicy*, the practice of examining sheep entrails for divination. Haruspices are described in more detail in §5.

## 3.2 General Running OVIS

Executables for the sheep, shepherd, baron, and the associated database effector are in `/path/to/ovisBuildDir/bin`. A quick command reference is shown below for the test example described in §7.2:

- Running the database effector:

```
./bin/ovis-db -d -t 16383 -u mysql://ovis@localhost/OVIS_Testone \
 -x /path/to/ovisSrcDir/data/testone.ovdb
```

- Running the shepherd:

```
./bin/shepherd --name=Testone \
  --database=mysql://ovis@localhost/OVIS_Testone
```

- Running the sheep:

```
./bin/sheep --name=Testone
```

- Running the baron:

```
./bin/baron
```

Note that you do not always have to start all components. If you are only taking data and not examining it, you only need to start the sheep and shepherd. If you are only examining data in an existing database and are not taking any additional data, you only need to start the shepherd and the baron. If you only want to look at the geometry of a cluster and neither want to take data nor perform any analyses, you only need to start the baron.

# 4 Setup

This section describes set up of the configuration files, samplers, and database necessary for running OVIS.

## 4.1 XML file

The XML file, canonically named `<clustername>.ovdb`, contains information on the cluster components, including additional components such as storage elements; their physical configuration; the metrics to be sampled; and IP addresses of the sheep, shepherd, and baron nodes.

The OVIS 2.0 release comes with an example XML file for the Whitney cluster, `whitneydemo.ovdb`. In most cases, it may be easiest to copy this file and edit it for your cluster. Metric data for this example is also included and will be discussed in §7.1.

An additional file, `whitneyterascalademo.ovdb` is included that illustrates use of OVIS for simultaneous monitoring of compute and storage resources. Finally, two small example files, `testone.ovdb` and `testonecpu.ovdb`, are included and discussed in §7.2 for use on your machine. The latter file also illustrates the specification for display of per-core data for multicore processors as discussed in §4.3.1.

This section describes items of note in the XML file.

### 4.1.1 Details of the XML file

In the *cluster* tag, the *name* field should be replaced with your cluster name.

The *component_types* section lists information for each type of component you have in your cluster, e.g., node, rack, switch. Each type should be listed separately in its own *component_type* block. Each component type should specify the following:

- type information - including the name by which the component will be identified, e.g., "rack" and whether it is a container or not, e.g., a rack will contain nodes and is therefore a container.
- drawing information that will represent that item's appearance (e.g., `render/Rack.vtp`, where the path is relative to the directory the ovdb file is in)
- size and slot information for drawing and the latter for determining the position of contained components if this component is a container. Units are in millimeters and we take 1 rack unit (1 RU or 1U) to be exactly 42 [*mm*] for convenience.
- sampler information - samplers that will run on this type of component, which metrics they will sample, and their interval. Samplers will be described in more detail in §4.3.

```
<component_types>
  <component_type
    type="rack" category="rack" is_container="1"
    glyph_shape="render/Rack.vtp"
    manufacturer="ASR"
    make="3100"
    model="ASR"
    description="TLCC cabinet"
    slot_origin="57.775,35,19"
    slot_size="500,700,44"
    slots_per_axis="1,1,42"
    slot_frame="1,0,0,  0,1,0,  0,0,1"
    height="1911" width="600" depth="737">
```

Figure 1: Excerpt from the Whitney XML file pertaining to the rack.

```
<component_type
  type="node" category="node" is_container="0"
  glyph_shape="render/RackNode1UFlat.vtp"
  glyph_image="render/Appro1ULight.vti"
  manufacturer="Appro"
  make="Supermicro"
  model="H8QM8"
  description="Whitney compute node, Quad-Core AMD Opteron Processor 8354, 32XGB RAM"
  width="550" depth="600" height="44">

   <!-- Metrics -->

  <!-- Sampler for position metrics -->
  <sampler name="ovMetricNodePositionSampler" interval="-1">
    <metric name="PositionX"         units="m"         storage_type="float" frequency="once"
onstancy="time"/>
    <metric name="PositionY"         units="m"         storage_type="float" frequency="once"
onstancy="time"/>
    <metric name="PositionZ"         units="m"         storage_type="float" frequency="once"
onstancy="time"/>
  </sampler>

  <!-- lm sensors sampler -->
  <sampler name="ovMetricLinuxTLCCLmsensorsSampler" interval="5">
   <metric name="CPU1_Temp" units="Celsius" storage_type="float" frequency="regular"/>
   <metric name="CPU2_Temp" units="Celsius" storage_type="float" frequency="regular"/>
   <metric name="CPU3_Temp" units="Celsius" storage_type="float" frequency="regular"/>
   <metric name="CPU4_Temp" units="Celsius" storage_type="float" frequency="regular"/>
   <metric name="SYS_Temp" units="Celsius" storage_type="float" frequency="regular"/>
   <metric name="VCoreA" units="V" storage_type="float" frequency="regular"/>
   <metric name="VCoreB" units="V" storage_type="float" frequency="regular"/>
   <metric name="VCoreC" units="V" storage_type="float" frequency="regular"/>
   <metric name="VCoreD" units="V" storage_type="float" frequency="regular"/>
   <metric name="3p3V" units="V" storage_type="float" frequency="regular"/>
   <metric name="5V" units="V" storage_type="float" frequency="regular"/>
   <metric name="12V" units="V" storage_type="float" frequency="regular"/>
   <metric name="5VSB" units="V" storage_type="float" frequency="regular"/>
   <metric name="VBATA" units="V" storage_type="float" frequency="regular"/>
   <metric name="FAN1" units="RPM" storage_type="float" frequency="regular"/>
   <metric name="FAN2" units="RPM" storage_type="float" frequency="regular"/>
   <metric name="FAN3_CPU2" units="RPM" storage_type="float" frequency="regular"/>
   <metric name="FAN4_CPU1" units="RPM" storage_type="float" frequency="regular"/>
   <metric name="FAN5" units="RPM" storage_type="float" frequency="regular"/>
   <metric name="FAN6" units="RPM" storage_type="float" frequency="regular"/>
   <metric name="FAN7_CPU4" units="RPM" storage_type="float" frequency="regular"/>
   <metric name="FAN8_CPU3" units="RPM" storage_type="float" frequency="regular"/>
   <metric name="FAN9" units="RPM" storage_type="float" frequency="regular"/>
  </sampler>
```

Figure 2: Excerpt from the Whitney XML file pertaining to the node.

Selections of these blocks are shown in Figure 1 for the rack, which is a container with slots, and in Figure 2 for the node, which has samplers. The Whitney cluster consists of nine racks; 284 compute nodes; twelve gateway nodes in Rack 9; and 4 login nodes, two of height 2U and two of height 4U in Rack 1. This layout can be see in the OVIS 2.0 physical display in many figures in §7.1.

Include in this section at least one option for the shepherd node. The shepherd node will be used for interactions with the database when taking data and/or when performing analyses (You do not always have to do both as described in §3.2). A shepherd node may also be a sheep (i.e., running a sampler) node.

The *instances* section lists the number of each type of component. The order in which these occur will determine the overall component identifications – CompId in database tables (discussed in §4.2.1) – and will be significant in particular for remote samplers (discussed in §4.3).

Multiple simultaneous shepherds are currently not a supported feature (This is discussed in §8.2). However, you can specify multiple shepherd possibilities in advance, thus allowing yourself options as to where you will choose to start a shepherd at a later time. This can either take the form of specifying multiple possible instances – in this example this would take the form of increasing the number of instances of Component Type "ts" in Figure 3 (top) and assigning different address information to each of those instances – or of assigning multiple addresses to the same instance (i.e., the same CompId). The use of address infomation is described below.

The *associations* section specifies the physical layout. Containers and contained components and their relative associations are specified. Containers must specify their type, their instance (by "num"), their overall position and orientation in space, and, optionally a short name by which they can be identified. Contained components must specify their type, their instance, the slot in the container in which they reside (number of slots were specified in the *component_type* block, and an optional short name (e.g., for Whitney node 1, "wn0").

Contained components can also be used to illustrate components of a node – e.g., fans or cores. Figure 4 illustrates the use of contained components for a multi-core display. The sampler for this case is described in §4.3.1.

Not all components in the *instances* must be in the *associations* section – for instance, you may not want to explicitly draw the shepherd nodes. Selections of these blocks are in Figure 3 (top).

The *addresses* section gives IP and MAC addresses by which sheep and shepherds are identified. An excerpt of this section is in Figure 3 (bottom). While Avahi is used by shepherds to advertise their availability, each component must have a unique identifier in order to insert information in the database. These identifiers appear in the CompId column of the **StartupData** table. You may specify addresses explicitly in the *addresses* section or you may use a regular expression to transform values from a column in the **StartupData** table into a component id (CompId) with a *hint*. Explicit address entries look like either of the two options below in the XML file:

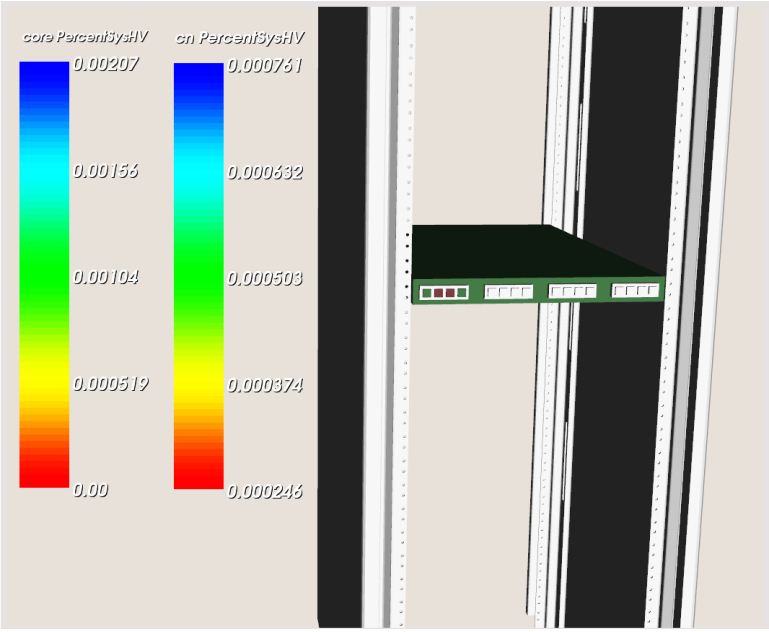
```
<addresses>
```

```
<!-- ** Instance information (how many components of each type, and how are they numbered?) ** -->
<instances>
  <components type="rack"   numbered="1-9"/>            <!-- Whitney Racks -->
  <components type="nodefouru" numbered="1-2"/>         <!-- Whitney fouru login nodes -->
  <components type="nodetwou" numbered="1-2"/>          <!-- Whitney twou login nodes -->
  <components type="node" numbered="1-300"/>             <!-- Whitney compute nodes -->
  <components type="ts" numbered="1"/>                   <!-- Whitney shepherd nodes -->
</instances>

<associations>
<!-- ** Component Association Data (containment,connectivity) ** -->
  <association label="physical" type="containment">
    <!--Rack 1 -->
    <container type="rack" num="1" position="0,0,0" orientation="1,0,0, 0,1,0, 0,0,1" name="rack1">
        <component type="nodefouru" num="1" slot="0,0,32" name="wn0"/>
        <component type="nodefouru" num="2" slot="0,0,28" name="wn1"/>
        <component type="nodetwou" num="1" slot="0,0,24" name="wn2"/>
        <component type="nodetwou" num="2" slot="0,0,22" name="wn3"/>
        <component type="node" num="1" slot="0,0,19" name="wn4"/>
        <component type="node" num="2" slot="0,0,18" name="wn5"/>
        <component type="node" num="3" slot="0,0,17" name="wn6"/>
        <component type="node" num="4" slot="0,0,16" name="wn7"/>
        <component type="node" num="5" slot="0,0,15" name="wn8"/>
        <component type="node" num="6" slot="0,0,14" name="wn9"/>
        <component type="node" num="7" slot="0,0,13" name="wn10"/>
        <component type="node" num="8" slot="0,0,12" name="wn11"/>
        <component type="node" num="9" slot="0,0,11" name="wn12"/>
        <component type="node" num="10" slot="0,0,10" name="wn13"/>
        <component type="node" num="11" slot="0,0,9" name="wn14"/>
        <component type="node" num="12" slot="0,0,8" name="wn15"/>
        <component type="node" num="13" slot="0,0,7" name="wn16"/>
        <component type="node" num="14" slot="0,0,6" name="wn17"/>
        <component type="node" num="15" slot="0,0,5" name="wn18"/>
        <component type="node" num="16" slot="0,0,4" name="wn19"/>
        <component type="node" num="17" slot="0,0,3" name="wn20"/>
        <component type="node" num="18" slot="0,0,2" name="wn21"/>
        <component type="node" num="19" slot="0,0,1" name="wn22"/>
        <component type="node" num="20" slot="0,0,0" name="wn23"/>
    </container>
    <!--Rack 2 -->
    <container type="rack" num="2" position="-650,0,0" orientation="1,0,0, 0,1,0, 0,0,1" name="rack2">
        <component type="node" num="21" slot="0,0,35" name="wn24"/>
        <component type="node" num="22" slot="0,0,34" name="wn25"/>
        <component type="node" num="23" slot="0,0,33" name="wn26"/>
        <component type="node" num="24" slot="0,0,32" name="wn27"/>
        <component type="node" num="25" slot="0,0,31" name="wn28"/>
        <component type="node" num="26" slot="0,0,30" name="wn29"/>
        <component type="node" num="27" slot="0,0,29" name="wn30"/>
        <component type="node" num="28" slot="0,0,28" name="wn31"/>
        <component type="node" num="29" slot="0,0,27" name="wn32"/>
        <component type="node" num="30" slot="0,0,26" name="wn33"/>
        <component type="node" num="31" slot="0,0,25" name="wn34"/>
        <component type="node" num="32" slot="0,0,24" name="wn35"/>
```

```
<!-- ** Address Data ** -->
<!-- Note: these addresses have all been changed from the real values -->
<addresses>
  <!-- shepherd -->
  <address compid="314" type="ipv4"     data="c0a85e01"/>
  <address compid="314" type="ethernet" data="AAAAAAAAAAAA"/>
  <!-- whitney nodes -->
  <address compid="14"    type="ipv4"     data="0a005005"/>
  <address compid="15"    type="ipv4"     data="0a005006"/>
  <address compid="16"    type="ipv4"     data="0a005007"/>
  <address compid="17"    type="ipv4"     data="0a005008"/>
  <address compid="18"    type="ipv4"     data="0a005009"/>
  <address compid="19"    type="ipv4"     data="0a00500a"/>
  <address compid="20"    type="ipv4"     data="0a00500b"/>
  <address compid="21"    type="ipv4"     data="0a00500c"/>
  <address compid="22"    type="ipv4"     data="0a00500d"/>
  <address compid="23"    type="ipv4"     data="0a00500e"/>
  <address compid="24"    type="ipv4"     data="0a00500f"/>
  <address compid="25"    type="ipv4"     data="0a005010"/>
  <address compid="26"    type="ipv4"     data="0a005011"/>
  <address compid="27"    type="ipv4"     data="0a005012"/>
  <address compid="28"    type="ipv4"     data="0a005013"/>
  <address compid="29"    type="ipv4"     data="0a005014"/>
  <address compid="30"    type="ipv4"     data="0a005015"/>
  <address compid="31"    type="ipv4"     data="0a005016"/>
  <address compid="32"    type="ipv4"     data="0a005017"/>
```

Figure 3: Excerpts from the Whitney XML file pertaining to the instances and associations (top) and addresses (bottom). The relationship between compid, component type, and component num ensures that all information for a given component is properly associated.




```
<association label="physical" type="containment">
  <container type="rack" num="1" position="0,0,0" orientation="1,0,0, 0,1,0, 0,0,1">
    <container type="cn" num="1" slot="0,0,23" name="e0">
      <container type="cpu" num="1" slot="0,0,0" name="e0c0">
        <component type="core" num="1" slot="0,0,0" name="e0c0c0"/>
        <component type="core" num="2" slot="1,0,0" name="e0c0c1"/>
        <component type="core" num="3" slot="2,0,0" name="e0c0c2"/>
        <component type="core" num="4" slot="3,0,0" name="e0c0c3"/>
      </container>
      <container type="cpu" num="2" slot="1,0,0" name="e0c1">
        <component type="core" num="5" slot="0,0,0" name="e0c1c0"/>
        <component type="core" num="6" slot="1,0,0" name="e0c1c1"/>
        <component type="core" num="7" slot="2,0,0" name="e0c1c2"/>
        <component type="core" num="8" slot="3,0,0" name="e0c1c3"/>
      </container>
      <container type="cpu" num="3" slot="2,0,0" name="e0c2">
        <component type="core" num="9" slot="0,0,0" name="e0c2c0"/>
        <component type="core" num="10" slot="1,0,0" name="e0c2c1"/>
        <component type="core" num="11" slot="2,0,0" name="e0c2c2"/>
        <component type="core" num="12" slot="3,0,0" name="e0c2c3"/>
      </container>
      <container type="cpu" num="4" slot="3,0,0" name="e0c3">
        <component type="core" num="13" slot="0,0,0" name="e0c3c0"/>
        <component type="core" num="14" slot="1,0,0" name="e0c3c1"/>
        <component type="core" num="15" slot="2,0,0" name="e0c3c2"/>
        <component type="core" num="16" slot="3,0,0" name="e0c3c3"/>
      </container>
    </container>
  </container>
</association>
```

Figure 4: Display (top) and assocation specification (bottom) of contained components used to illustrate a node with four CPUs, each of which has four cores. The association excerpt is from the `testonecpu.ovdb` file.

```
  <address compid="2" type="ipv4"     data="481d53a4"/>
  <address compid="2" type="ethernet" data="0017deadbeef"/>
</addresses>

<addresses>
  <address comptype="cn" compnum="1" type="ipv4"     data="481d53a4"/>
  <address comptype="cn" compnum="2" type="ipv4"     data="481d53a5"/>
</addresses>
```

where the type attribute should be either "ipv4" (for TCP/IP addresses) or "ethernet" (for MAC addresses) and the data attribute should be a hexadecimal number that is either 4 or 6 bytes long depending on the address type.

Hints are specified like so:

```
<addresses>
  <hint>
    <key column="NodeName">
      <regex comptype="cn" compnum="%1">cn([0-9]+)</regex>
      <regex comptype="cn" compnum="1">admin</regex>
    </key>
  </hint>
</addresses>
```

Each *hint* tag (there may be several) must always contain exactly one *key* tag. The column attribute of the *key* tag specifies a column in the **StartupData** table. Note that when the sheep program is run on a component without an entry in **StartupData**, the sheep will run a special metric sampler that adds columns to **StartupData** containing the node's name (NodeName in the example above), operating system, kernel version, and other information reported by the `uname` command. This results in many rows in **StartupData** with network addresses and other information specified but no values in the CompId column that assigns a unique id to the component by its network address. You may use *regex* tags in the XML file to assign a component number to any entry in **StartupData** by extracting a numeric substring from the key column's value with a regular expression. In the example above, any entry in the NodeName column that starts with "cn" and ends with a decimal number will be assigned the CompId corresponding to the "cn" CompType and whose *instance* number matches the number in the hostname. It is also possible to create regular expressions that match only a given hostname (such as the second regular expression in the example) and specify both the component type and number directly. Note that component *numbers* are unique across all components of a given type while component *ids* are unique across all components of all types. The **ComponentTable** contains a mapping between component numbers and ids. If you use hints to specify component numbers (and thus ids), you must allow the sheep processes to insert their network addresses, host names, and other data into the **StartupData** table and then run `ovis-db` with `-t 512` (c.f. §4.2) to assign values to the CompId column in **StartupData**. Only after you do this will the sheep be able to insert metric data into the database.

```
<!--Rack 10  (Terascala) -->
<container type="rack" num="10" position="-7000,0,0" orientation="1,0,0, 0,1,0, 0,0,1" name="terascala1">
  <!-- narnia1 is the mds, all others are oss -->
  <container type="tschassis" num="1" slot="0,0,0" name="TS21">
    <component type="tsnode" num="1" slot="0,0,0" name="narnia1"/>
    <component type="tsnode" num="2" slot="1,0,0" name="narnia2"/>
    <component type="tsnode" num="3" slot="2,0,0" name="narnia3"/>
    <component type="tsnode" num="4" slot="3,0,0" name="narnia4"/>
    <component type="tsnode" num="5" slot="4,0,0" name="narnia5"/>
  </container>
  <container type="tschassis" num="2" slot="0,0,8" name="TS22">
    <component type="tsnode" num="6" slot="0,0,0" name="narnia6"/>
    <component type="tsnode" num="7" slot="1,0,0" name="narnia7"/>
    <component type="tsnode" num="8" slot="2,0,0" name="narnia8"/>
    <component type="tsnode" num="9" slot="3,0,0" name="narnia9"/>
    <component type="tsnode" num="10" slot="4,0,0" name="narnia10"/>
  </container>
  <container type="tschassis" num="3" slot="0,0,16" name="TS23">
    <component type="tsnode" num="11" slot="0,0,0" name="narnia11"/>
    <component type="tsnode" num="12" slot="1,0,0" name="narnia12"/>
    <component type="tsnode" num="13" slot="2,0,0" name="narnia13"/>
    <component type="tsnode" num="14" slot="3,0,0" name="narnia14"/>
    <component type="tsnode" num="15" slot="4,0,0" name="narnia15"/>
  </container>
  <container type="tschassis" num="4" slot="0,0,24" name="TS24">
    <component type="tsnode" num="16" slot="0,0,0" name="narnia16"/>
    <component type="tsnode" num="17" slot="1,0,0" name="narnia17"/>
    <component type="tsnode" num="18" slot="2,0,0" name="narnia18"/>
    <component type="tsnode" num="19" slot="3,0,0" name="narnia19"/>
    <component type="tsnode" num="20" slot="4,0,0" name="narnia20"/>
  </container>
```

Figure 5: Excerpt from the Whitney-Terascala XML file with the association information for the Terascala storage rack.

Note that OVIS can simultaneously monitor any number and type of resources. The example file `whitneyterascalademo.ovdb` includes specification information for the Whitney cluster and an associated Terascala [6] Storage rack which is a container of 4 chassis each of which contains 5 blades. The *associations* section of this file specifying this rack is shown in Figure 5. The resultant OVIS physical display for the combined resources is shown in Figure 6.

## 4.2 Database Effector

Once you have a properly formatted ovdb XML file (located, say, at `/path/to/cluster.ovdb`), you should run the `ovis-db` program to populate a database for the sheep, shepherds, and baron to use given the XML file. The `ovis-db` utility also allows you to insert test data (formatted as an XML ovdata file) into a database but that is not covered here.

First, you should create the database. The name of the database must be OVIS_⟨*ClusterName*⟩ where ⟨*ClusterName*⟩ matches the *name* attribute of the *cluster* tag of your ovdb file. We will assume ⟨*ClusterName*⟩ is "Cluster" for the rest of this section. For MySQL, run this command to create the database

```
echo "CREATE DATABASE OVIS_Cluster;" | mysql -u ovis
```

For PostgreSQL, run this command

```
createdb -U ovis OVIS_Cluster
```

Once the database exists, run `ovis-db` to populate it. For MySQL databases use:

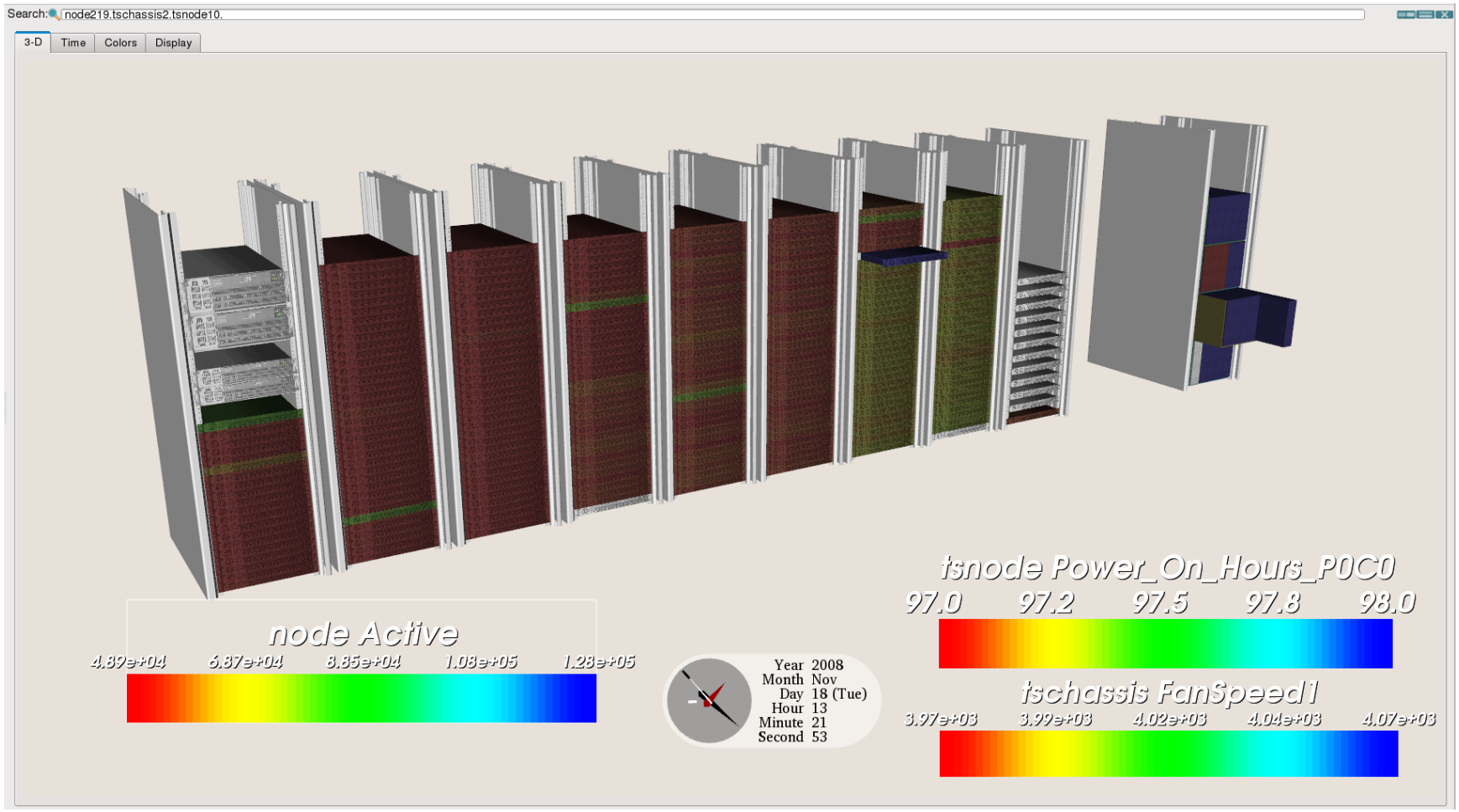



Figure 6: Physical display of the Whitney compute cluster nodes and the Terascala Storage Rack. For the compute nodes, Active Memory is displayed; for the Terascala Chassis, Fan Speed is displayed; for the Terascala Blades, Power On Hours are displayed. One instance of each of these component types is popped out in the figure. Metric values for the Terascala nodes are being obtained via remote samplers running on the Terascala admin node (not shown).

| Constant | Action |
|---|---|
| 1 | Create the database |
| 2 | Create tables |
| 4 | Load dynamic library functions into the database server |
| 8 | Populate ComponentTypes table |
| 16 | Populate MetricValueTypes table |
| 32 | Populate MetricValueTableIndex table |
| 64 | Populate Components table |
| 128 | Populate MetricCollectionSamplers table |
| 256 | Populate RemoteSamplerLookup table |
| 512 | Populate StartupData table |
| 1024 | Populate ComponentGlyphs table |
| 2048 | Prepare trigger actions |
| 4096 | Populate metric tables with static data (e.g. component coordinates) |

Table 1: Numeric constants that may be summed and passed to the `ovis-db`'s `-t` flag specifying which actions are to be performed.

```
ovis-db -d -t 8191 -x /path/to/cluster.ovdb \
  -u mysql://ovis@localhost/OVIS_Cluster
```

For PostgreSQL databases use:

```
ovis-db -d -t 8191 -x /path/to/cluster.ovdb \
  -u psql://ovis@localhost/OVIS_Cluster
```

The `-d` flag tells `ovis-db` to drop any existing rows or tables as necessary. The `-t 8191` option instructs `ovis-db` which actions to perform. The number specified is a bit vector composed by adding numbers from Table 1. The actions are executed in the order they appear in the table. Generally you will only need to run this command once as specified above.

While most actions simply create or populate tables, some perform less trivial tasks and order is important. The dynamic libraries (`libovis-mysql.so` and/or `libovis-psql.so`) must be installed into their proper locations before `ovis-db` is asked to have the SQL server process load them. The SQL server must have loaded these dynamic libraries before the database triggers are prepared since the triggers invoke functions provided by these libraries[2]. If you load the example Whitney mysqldump for your first test case, rather than creating the database tables via the effector as described above, it is recommended that you run the effector with flag `-t 4` in order to ensure that this function is loaded, in case it is not loaded as part of loading the mysqldump.

[2]Specifically, the function `signal_local_haruspex` is provided to send a UDP message to the local shepherd when rows of the HaruspexRequests or haruspex subresults tables discussed later are inserted or modified.

### 4.2.1 Database Tables

Information in the XML file is used to set up the database tables. Of particular interest are:

- **ComponentTypes** - information about the Component Types
- **ComponentTable** - associates a given Component Type and number with the unique CompId and its name.
- **StartupData** - for each component address information and samplers
- **EventIndex** - unique identification of metric value entries including in which table the entry was stored and where in that table the entry is stored (MetricValue tables are described in more detail below).
- **TimeIndex** - association of the unique identifiers in the ovdbtabEventIndex and their time of occurrence.

You should also be aware of the **HaruspexIds** table. This table assigns unique instance numbers to any active shepherds on this database. The use of "Haruspex" in the name reflects the intent that analyses may be performed differently if there are multiple shepherds available. Multiple shepherds is not a supported feature at this time and is discussed (briefly) in §8.2.

Database tables for the samplers are described in more detail in §4.3. Database tables for analyses are described in §5.2.

## 4.3 Data Samplers

The OVIS 2.0 release comes with several samplers, some of which read from well-known locations on Linux installations and will probably work as is, and some which will have to be altered to work with your system. The former include those that read from `/proc`, such as `ovMetricLinuxProcStatUtilSampler` and `ovMetricLinuxProcMemInfoSampler`. The latter will require some modification of configuration information, such as available metrics from the source or path information as in, for example, `ovMetricExampleLinuxlmsensorsSampler` and `ovMetricExampleLinuxSmartctlSampler`.

Samplers can either be *local* or *remote*. A local sampler is a process running on machine *A* that collects metrics related to machine *A* and then inserts them on behalf of itself into an OVIS database – which is usually some remote machine *C* but could also be *A*. A remote sampler is a process running on machine *A* that collects metrics related to some other machine *B* and then inserts them on behalf of machine *B* into an OVIS database – which is usually some remote machine *C* but could be *B* (or even *A*, but this would be odd). Examples of remote samplers are `ovMetricExampleLinuxIMPItoolSampler` and `ovMetricExampleLinuxSNMPNodeSampler`. In

the case of the combined Whitney-Terascala monitoring, all of the Terascala samplers were remote with chassis data being collected via an SNMP sampler, and blade and associated disk data being collected via both an SNMP sampler and a Smartctl sampler. These remote samplers were running on the Terascala admin node. Of particular note are the two versions of the `ovMetricLinuxProcStatUtil` Sampler, which comes in both local and remote versions, where usage is dependent on the fidelity of the component specification as described in §4.3.1.

```
ovExampleLinuxlmsensorsMetric ClusterlmsensorsMetricTemplate[ovMetricExampleLinuxlmsensorsSampler::NumMetrics] = {
  /* metric name                          , ovis name,   stride, style, flag, lastval */
  { std::string("CPU1 Temp"),          std::string("CPU1_Temp"),    1, ovis::ALARM,    0,  0.},
  { std::string("CPU2 Temp"),          std::string("CPU2_Temp"),    1, ovis::ALARM,    0,  0.},
  { std::string("CPU3 Temp"),          std::string("CPU3_Temp"),    1, ovis::ALARM,    0,  0.},
  { std::string("CPU4 Temp"),          std::string("CPU4_Temp"),    1, ovis::ALARM,    0,  0.},
  { std::string("SYS Temp"),           std::string("SYS_Temp"),     1, ovis::ALARM,    0,  0.},
  { std::string("VCoreA"),             std::string("VCoreA"),       1, ovis::ALARM,    0,  0.},
  { std::string("VCoreB"),             std::string("VCoreB"),       1, ovis::ALARM,    0,  0.},
  { std::string("VCoreC"),             std::string("VCoreC"),       1, ovis::ALARM,    0,  0.},
  { std::string("VCoreD"),             std::string("VCoreD"),       1, ovis::ALARM,    0,  0.},
  { std::string("+3.3V"),              std::string("3p3V")       ,  1, ovis::ALARM,    0,  0.},
  { std::string("+5V"),                std::string("5V")         ,  1, ovis::ALARM,    0,  0.},
  { std::string("+12V"),               std::string("12V")        ,  1, ovis::ALARM,    0,  0.},
  { std::string("5VSB"),               std::string("5VSB")       ,  1, ovis::ALARM,    0,  0.},
  { std::string("VBATA"),              std::string("VBATA")     ,  1, ovis::ALARM,    0,  0.},
  { std::string("FAN1"),               std::string("FAN1")       ,  1, ovis::ALARM,    0,  0.},
  { std::string("FAN2"),               std::string("FAN2")       ,  1, ovis::ALARM,    0,  0.},
  { std::string("FAN3/CPU2"),          std::string("FAN3_CPU2") ,  1, ovis::ALARM,    0,  0.},
  { std::string("FAN4/CPU1"),          std::string("FAN4_CPU1") ,  1, ovis::ALARM,    0,  0.},
  { std::string("FAN5"),               std::string("FAN5")       ,  1, ovis::ALARM,    0,  0.},
  { std::string("FAN6"),               std::string("FAN6")       ,  1, ovis::ALARM,    0,  0.},
  { std::string("FAN7/CPU4"),          std::string("FAN7_CPU4") ,  1, ovis::ALARM,    0,  0.},
  { std::string("FAN8/CPU3"),          std::string("FAN8_CPU3") ,  1, ovis::ALARM,    0,  0.},
  { std::string("FAN9"),               std::string("FAN9")       ,  1, ovis::ALARM,    0,  0.}
};
```

Figure 7: Excerpt from the lmsensors sampler illustrating specification of metric name, stride, data type etc.

In the sampler, each metric must be identified by its name, the frequency of that metric's insertion into the database (the stride), and the metric value's data type (e.g., `ovis::INT`), as illustrated in Figure 7. Note that some metrics lend themselves to reporting as instantaneous values while others as differences from their last value, as for counters. The sampler inserts the metric data value into the database via its `RecordXXX` (e.g., `RecordInt`) function that takes the data value and a number identifier that uniquely distinguishes the component and the metric. Required functions in the sampler, such as `GetMetricName` and `GetMetricStorageType`, associate a particular number identifier with a particular metric. For local samplers, unique distinguishing of the component is done innately by the identity of the sheep. The situation is more complex for remote samplers, however. Not only must the remote sampler associate a particular numeric identifier with a particular metric, but the numeric identifier must also be used to distinguish the component. For example, a remote sampler that keeps track of 3 metrics on behalf of 5 other components will use 15 unique identifiers. The mapping of these identifiers to the particular components is specified via the sampler listing in the XML file and the `metric node map`, as described below. These identifiers are then placed into the **RemoteSamplerLookup** table of the database by the `ovis-db` command and read by the sheep process running the remote sampler at startup.

Setup in the XML file for remote samplers involves the following:

- you must specify dummy samplers with the metrics of the correct name for the component on whose behalf data will be taken, in order for the correct tables to be established;

```
<component_type
  type="tschassis" category="blade chassis" is_container="1"
  glyph_shape="render/ts_chassis.vtp"
  manufacturer="Terascala"
  make=""
  model=""
  description="Terascala chassis using Force10 chassis description"
  slot_origin="35,-10,11"
  slot_size="88,600,352"
  slots_per_axis="8,1,1"
  slot_frame="1,0,0,  0,1,0,  0,0,1"
  height="352" width="530" depth="700">

  <!-- Dummy sampler for some dummy metrics for these dumb chassis  -->
  <sampler name="ovMetricSwitchInfoSampler" interval="-1">
    <metric name="Position"          units="m"          storage_type="posn3" frequency="once"/>
    <metric name="Orientation"       units="m"          storage_type="vec3"  frequency="once"/>
  </sampler>

  <!-- SNMP Chassis sampler (dummy so that tables are
  created. this is actually running on y node) -->
    <sampler name="ovMetricTerascalaSNMPChassisDummySampler" interval="10000">
      <metric name="FanSpeed1"    units="RPM"    storage_type="int"    frequency="regular"/>
      <metric name="FanSpeed2"    units="RPM"    storage_type="int"    frequency="regular"/>
      <metric name="FanSpeed3"    units="RPM"    storage_type="int"    frequency="regular"/>
      <metric name="FanSpeed4"    units="RPM"    storage_type="int"    frequency="regular"/>
      <metric name="FanSpeed5"    units="RPM"    storage_type="int"    frequency="regular"/>
      <metric name="FanSpeed6"    units="RPM"    storage_type="int"    frequency="regular"/>
      <metric name="Temp"         units="Celsius"    storage_type="int"    frequency="regular"/>
    </sampler>

</component_type>
```

```
<component_type
  type="tsadmin" category="node" is_container="0"
  manufacturer="Dell"
  make="PowerEdge"
  model="1850"
  description="terascala admin node"
  width="550" depth="600" height="44">

<!-- ** Metrics ** -->

  <!-- Just a signal that the machine is alive -->
  <sampler name="ovMetricHeartbeat" interval="10000">
    <metric name="Heartbeat" units="1" storage_type="none" frequency="regular"/>
  </sampler>


  <!-- SNMP Node sampler (remote sampler for the nodes) -->
    <sampler name="ovMetricTerascalaSNMPNodeSampler" interval="30">
      <metric name="CPU1CoreTemp"    units="Celsius"    storage_type="int"    frequency="regular"/>
      <metric name="AmbientTemp"    units="Celsius"    storage_type="int"    frequency="regular"/>
      <metric name="BladeTemp1"    units="Celsius"    storage_type="int"    frequency="regular"/>
      <metric name="BladeTemp2"    units="Celsius"    storage_type="int"    frequency="regular"/>
      <metric name="BladeTemp3"    units="Celsius"    storage_type="int"    frequency="regular"/>
      <metric name="BladeChassisTemp"    units="Celsius"    storage_type="int"    frequency="regular"/>
    </sampler>


  <!-- SNMP Chassis sampler (remote sampler for the chassis) -->
    <sampler name="ovMetricTerascalaSNMPChassisSampler" interval="30">
      <metric name="FanSpeed1"    units="RPM"    storage_type="int"    frequency="regular"/>
      <metric name="FanSpeed2"    units="RPM"    storage_type="int"    frequency="regular"/>
      <metric name="FanSpeed3"    units="RPM"    storage_type="int"    frequency="regular"/>
      <metric name="FanSpeed4"    units="RPM"    storage_type="int"    frequency="regular"/>
      <metric name="FanSpeed5"    units="RPM"    storage_type="int"    frequency="regular"/>
      <metric name="FanSpeed6"    units="RPM"    storage_type="int"    frequency="regular"/>
      <metric name="Temp"         units="Celsius"    storage_type="int"    frequency="regular"/>
    </sampler>


    <!-- SysBlockStat (now remotely for the nodes) -->
  <sampler name="ovMetricTerascalaSysBlockStatRemoteSampler" interval="100000">
    <metric name="sda_READ_IOs"                units="1"    storage_type="float"      frequency="regular"/>
    <metric name="sda_READ_MERGES"             units="1"    storage_type="float"      frequency="regular"/>
    <metric name="sda_READ_SECTORS"            units="1"    storage_type="float"      frequency="regular"/>
    <metric name="sda_READ_TICKS"              units="1"    storage_type="float"      frequency="regular"/>
    <metric name="sda_WRITE_IOs"               units="1"    storage_type="float"      frequency="regular"/>
    <metric name="sda_WRITE_MERGES"            units="1"    storage_type="float"      frequency="regular"/>
    <metric name="sda_WRITE_SECTORS"           units="1"    storage_type="float"      frequency="regular"/>
    <metric name="sda_WRITE_TICKS"             units="1"    storage_type="float"      frequency="regular"/>
```

Figure 8: Excerpts from the Whitney-Terascala XML file with partial specification of the remote samplers. The samplers (top) given for the component to which the data pertains, in this case the Chassis, are dummy and are just used to establish the metric tables. The samplers (bottom) for the component that will actually do the database insertion are real.

```
<association label="remotesamplers" type="sampler">
    <sampler_instance name="ovMetricTerascalaSNMPNodeSampler" type="tsadmin" num="1">
        <metric_node_map metric="0-5"     type="tsnode" num="001" />
        <metric_node_map metric="6-11"    type="tsnode" num="002" />
        <metric_node_map metric="12-17"   type="tsnode" num="003" />
        <metric_node_map metric="18-23"   type="tsnode" num="004" />
        <metric_node_map metric="24-29"   type="tsnode" num="005" />
        <metric_node_map metric="30-35"   type="tsnode" num="006" />
        <metric_node_map metric="36-41"   type="tsnode" num="007" />
        <metric_node_map metric="42-47"   type="tsnode" num="008" />
        <metric_node_map metric="48-53"   type="tsnode" num="009" />
        <metric_node_map metric="54-59"   type="tsnode" num="010" />
        <metric_node_map metric="60-65"   type="tsnode" num="011" />
        <metric_node_map metric="66-71"   type="tsnode" num="012" />
        <metric_node_map metric="72-77"   type="tsnode" num="013" />
        <metric_node_map metric="78-83"   type="tsnode" num="014" />
        <metric_node_map metric="84-89"   type="tsnode" num="015" />
        <metric_node_map metric="90-95"   type="tsnode" num="016" />
        <metric_node_map metric="96-101"  type="tsnode" num="017" />
        <metric_node_map metric="102-107" type="tsnode" num="018" />
        <metric_node_map metric="108-113" type="tsnode" num="019" />
        <metric_node_map metric="114-119" type="tsnode" num="020" />
    </sampler_instance>
    <sampler_instance name="ovMetricTerascalaSNMPChassisSampler" type="tsadmin" num="1">
        <metric_node_map metric="0-6"   type="tschassis" num="001" />
        <metric_node_map metric="7-13"  type="tschassis" num="002" />
        <metric_node_map metric="14-20" type="tschassis" num="003" />
        <metric_node_map metric="21-27" type="tschassis" num="004" />
    </sampler_instance>

    <sampler_instance name="ovMetricTerascalaSysBlockStatRemoteSampler" type="tsadmin" num="1">
        <metric_node_map metric="0-32"      type="tsnode" num="001" />
        <metric_node_map metric="33-65"     type="tsnode" num="002" />
        <metric_node_map metric="66-98"     type="tsnode" num="003" />
        <metric_node_map metric="99-131"    type="tsnode" num="004" />
        <metric_node_map metric="132-164"   type="tsnode" num="005" />
        <metric_node_map metric="165-197"   type="tsnode" num="006" />
        <metric_node_map metric="198-230"   type="tsnode" num="007" />
        <metric_node_map metric="231-263"   type="tsnode" num="008" />
        <metric_node_map metric="264-296"   type="tsnode" num="009" />
        <metric_node_map metric="297-329"   type="tsnode" num="010" />
        <metric_node_map metric="330-362"   type="tsnode" num="011" />
        <metric_node_map metric="363-395"   type="tsnode" num="012" />
        <metric_node_map metric="396-428"   type="tsnode" num="013" />
        <metric_node_map metric="429-461"   type="tsnode" num="014" />
        <metric_node_map metric="462-494"   type="tsnode" num="015" />
```

Figure 9: Excerpt from Whitney-Terascala XML file showing the `metric node maps` which associate remote sampler metric numbering and the corresponding components.

- you must specify real remote samplers with the metrics of the correct name and the real collection interval on the component that is actually doing the collection and insertion; and

- you must specify a `metric node map` that associates a remote sampler instance and its unique metric identifiers with the component upon which it is running and the component on whose behalf data is being taken.

In Figure 8 the sampler specification for the Terascala chassis is a dummy, with the sampler specification on the Terascala admin node being the actual remote sampler that will be instantiated. The metric node map in the *associations* section shown in Figure 9 associates the unique metric numberings of the real sampler running on the Terascala admin node with the particular Terascala component (e.g., Chassis or Blade) to which those metrics pertain. For instance, in Figure 9 metrics 0-6 (the unique identifiers in the remote sampler) in the `ovMetricTerascalaSNMPChassisSampler` running on the first instance of type tsadmin (which is the only admin node) are actually values for the first chassis, tsnode 1 (the component types and number associations are in the *instances* and *associations* sections as described earlier with respect to the Whitney XML file). The order of the metrics (e.g., which metric is 0) is determined as previously described.

There is currently no mechanism to dynamically add in a sampler once the OVIS database has been instantiated, as the tables for the metric data are set up at that time. These are discussed in §4.3.2.

### 4.3.1 Local vs. Remote Samplers and Component Fidelity – Multicore Example

```
<sampler_instance name="ovMetricLinuxProcStatUtilRemoteSampler" type="cn" num="1">
  <metric_node_map metric="0-3"     type="cn"    num="1"  />
  <metric_node_map metric="4-7"     type="core"   num="1"  />
  <metric_node_map metric="8-11"     type="core"   num="2"  />
  <metric_node_map metric="12-15"     type="core"   num="3"  />
  <metric_node_map metric="16-19"     type="core"   num="4"  />
  <metric_node_map metric="20-23"     type="core"   num="5"  />
  <metric_node_map metric="24-27"     type="core"   num="6"  />
  <metric_node_map metric="28-31"     type="core"   num="7"  />
  <metric_node_map metric="32-35"     type="core"   num="8"  />
  <metric_node_map metric="36-39"     type="core"   num="9"  />
  <metric_node_map metric="40-43"     type="core"   num="10"  />
  <metric_node_map metric="44-47"     type="core"   num="11"  />
  <metric_node_map metric="48-51"     type="core"   num="12"  />
  <metric_node_map metric="52-55"     type="core"   num="13"  />
  <metric_node_map metric="56-59"     type="core"   num="14"  />
  <metric_node_map metric="60-63"     type="core"   num="15"  />
  <metric_node_map metric="64-67"     type="core"   num="16"  />
</sampler_instance>
```

Figure 10: Excerpt from the multicore processor XML file showing the `metric node map` which associates sampler metric numbering and the corresponding components. This case contains both node and core mappings.

In the storage case described in the previous section, the use of a remote sampler was required as data could only be obtained from the admin node. However, there are cases where both local and remote samplers are possibilities. For example, for multicore processors, core utilization information is obtained via `/proc/stat` on the node, but the sampler can be either local or remote depending on the fidelity of the display. That is, if the nodes are illustrated only to the node level, as they are in the Whitney example, then the sampler is a local sampler, collecting on the node with the intent of displaying that data on the node. However, if the nodes are illustrated to the core level as in Figure 4, then the sampler is a remote sampler, collecting on the node but with the intent of displaying that data on the cores. Of particular interest regarding the remote version of the sampler, `ovMetricLinuxProcStatUtilRemoteSampler`, is that the `/proc/stat` contains overall node cpu utilization as well. This sampler thus collects both node and core data, thereby functioning both locally and remotely, however, in order to handle the remote component mapping, it is a remote sampler. In this case the "Dummy" tables are not truly dummy, as they will hold the node data, but not the core data, and the `metric node map` must specify mapping information for both the node metrics and the core metrics. The `metric node map` for the remote sampler in the case of a node with four cpus and four cores per cpu as described in `testonecpu.ovdb` is shown in Figure 10, illustrating the specification of both node and core mapping. Further, note that in the local case (i.e., displayed on the node) the per core metrics must all have unique names per core (e.g., CPU0UserPercUtil, CPU1UserPercUtil) to distinguish to which core the metrics pertain, whereas in the remote case (i.e., displayed on the core) they do not, as they are uniquely identified by the core. One can further contrast the `ovMetricLinuxProcStatUtilSampler` and the `ovMetricLinuxProcStatUtilRemoteSampler` and associated `testone.ovdb` and `testonecpu.ovdb` files, respectively, to contrast the use of local and remote samplers for the same data.

### 4.3.2 Database Tables For Samplers

There are separate database tables for each combination of component type and metric that are canonically named **'Metric'** ***ComponentType MetricName*** **'Values'**. As an example, compute nodes (abbreviated "cn") with a metric named CPUTemp would have metric values stored in a table named **MetricCnCPUTempValues**. Each row corresponds to a unique reporting of a metric value and its associated component. When combined with information from the **EventIndex** and **TimeIndex**, one can determine which component reported what value at what time.

The **MetricValueTableIndex** holds information on the MetricTables, in particular associating a particular table by name with its identifier and the identifier of the sampler associated with that metric. The samplers are listed in the **MetricCollectionSamplers** table, along with information regarding upon which component type they sample on behalf of and on what interval they sample. The interval is determined at setup time from information in the XML file. With the information from these tables, one can determine into which table a particular metric is inserted, for which type of component, and by which sampler. This is of particular interest for the remote samplers where the component running the sampler is not the component for which the value actually pertains. Remote sampler association information thus requires information from the **RemoteSamplerLookup** table which associates a sampler, a metric, the component that is reporting the values into the database, and the component for which the value actually pertains.

Knowledge of these tables can help in debugging samplers, particularly remote samplers, as described in the next section.

### 4.3.3 Debugging Remote Samplers

Since the setting up of the remote samplers is non-trivial it is suggested that you verify the associations after you have instantiated the database with the Database Effector §4.2. In particular, note that the Database Effector will warn you if you put in a duplicate metric map, but there is nothing that tells you:

- if there are not enough or too many numbers compared to the number of overall metrics
- if there are the wrong number of metrics mapped onto a given remote component
  - this is to allow you to deliberately not collect metrics that you are not interested in, but it also opens you up to accidentally missing metrics, or accidentally numbering wrong
- if there is a bad type and/or number of the remote component in the metric map or in the addresses section
  - this could lead to the sampler not starting, or starting but not being able to write the metrics to the database properly

In order to verify the associations, check the **RemoteSamplerLookup** table (which matches up CompId, SamplerId, MetricId, and RemoteCompId) for the following things:

- cross-reference CompId and RemoteCompId in **RemoteSamplerLookup** with the **ComponentTable** to make sure that the component associations are correct.
- make sure that the CompId is not -1 or something you don't want it to be – if so you may have misnamed the sampler or the metric in the XML file
- cross-reference SamplerId in **RemoteSamplerLookup** with those in **MetricCollectionSamplers**, to make sure that the remote sampler identity is correct.
- make sure that the MetricIds are correct. In particular:
  - verify that there are the correct number of them
  - verify that the right ones go with the right remote CompId

Do not use hyphens in your metric names.

You can test samplers (both local and remote) via the following:

```
./bin/ovisTests testSampler -sampler mySampler interval iterations
```

where mySampler is the sampler to be tested and interval is the interval between iterations of the test. The test will print values to stdout and not to a database.

# 5 Haruspices

In Roman practice inherited from the Etruscans, a *haruspex* (plural *haruspices*) was a man trained to practice a form of divination called *haruspicy*, the inspection of the entrails of sacrificed animals, especially the livers of sacrificed sheep.

In this section, we first provide an overview of the analysis engines, which are called haruspices in OVIS parlance. We subsequently illustrate the utilization of these haruspices by providing an application of the multi-correlative haruspex; for more context about this application example, please refer to [3].

## 5.1 Overview

**Definition 5.1.** A OVIS 2.0 haruspex is a process that:

1. is triggered by the baron,
2. runs on a shepherd node,
3. executes an analytical engine.

In particular, haruspices rely on specific tables of the underlying OVIS 2.0 database.

There are currently four types of analyses supported by OVIS, although only three are part of the public release; only these are described here. Readers interested in the fourth type (Bivariate Bayesian) should contact the authors to discuss licensing options.

Each haruspex has the option to be run using either the learn or monitor modes of operation: In learn a *model* is calculated or inferred from unmodified data. Such a model can take several forms, such as statistical moment estimators, PDFs, *etc.*. In monitor the roles are here interchanged with those of the learn mode: the data is now assessed with respect to a given model. The output of the monitor mode is a collection of *outliers*, described in a way that allows for unambiguous and efficient retrieval of the particular components and times to which these correspond; the output may also be presented as an ordered list so as to reflect a gradation in severity or abnormality of behavior. The output may also be seen in the physical view, where components' values at the displayed time can be compared to the calculated model and colored accordingly. Note that reportable cases may occur either when a particular event diverges from the model more than what has been set as acceptable or because no (or fewer than specified) events of a particular type occurred. For instance, outliers – which may be defined in several ways depending on the type of model being used – can be identified as elements of the data set that deviate from what the model predicts within pre-defined acceptability bounds.

Within this framework, the currently available engines are the following:

**Descriptive haruspices:** In learn mode, descriptive statistics of the data set of interest are calculated (estimators of the mean, standard deviation, skewness, kurtosis, as well as bounds). These statistics can be interpreted directly by the user, or be used as input parameters to the monitor mode of the descriptive engine itself or even of another engine, e.g., to complement expert knowledge prior to Bayesian parameter estimation. In monitor mode, relative distance in terms of mean and standard deviation (which, as indicated, *may* be the result of a prior learn stage) is the criterion according to which outliers are detected, based on user-specified probabilistic thresholds.

**Bivariate and Multivariate Correlative haruspices:** The goal of these engine is to seek anomalous behaviors by calculating (in learn mode) or devising (with "expert knowledge") multivariate correlation statistics, via mean vectors and covariance matrices – and thus, implicitly, a multiple linear regression model – for a set of tuples of variables of interest, and examining (in monitor mode) how individual observations of these tuples of variables of interest deviate from the aforementioned model. Note that the bivariate haruspex explicitly presents the linear regression model to the user; the multivariate does not because multiple regressions can be calculated, resulting in user interface complications which are not handled yet. Such deviations are characterized in terms of the the multivariate Mahalanobis distance computed with the mean vector and covariance matrix. This is especially useful to prevent the user from conducting more advanced and costly analysis such as running a Bayesian engine when linear correlation between metrics can be evinced.

## 5.2 Haruspex Output Tables

This section is especially intended for the advanced OVIS user, as it delves into the details of "under the hood" of the database model used by haruspices. The user who will mostly use the Baron and does not plan on manually inspecting the haruspex tables can directly skip to §5.2.4 where reported events tables are described.

All haruspices report their results in output tables of the underlying OVIS database. This approach enables data persistence, thus allowing for later inspection of the results, storage, comparison, etc. Each haruspex uses output tables specific to it, as specified below using SQL types (note that the SequenceId field is unused in OVIS 2.0). These tables will likely change in future versions of OVIS.

### 5.2.1 Descriptive Haruspex-Specific Output Tables

In learn mode, each descriptive haruspex logs its results into the table **HaruspexDescriptiveSubResults** with primary key (RequestId, SequenceId, Rank):

| RequestId | SequenceId | Rank | SampleSize | Minimum | Maximum | Sum1 | Sum2 | Sum3 | Sum4 |
|---|---|---|---|---|---|---|---|---|---|
| INT(11) | INT(11) | INT(11) | INT(11) | DOUBLE | DOUBLE | DOUBLE | DOUBLE | DOUBLE | DOUBLE |

Subsequently, a MySQL trigger updates the summary table **HaruspexDescriptiveResults** with primary key (RequestId, SequenceId):

| RequestId | SequenceId | SampleSize | Minimum | Maximum | Sum1 | Sum2 | Sum3 | Sum4 |
|---|---|---|---|---|---|---|---|---|
| INT(11) | INT(11) | INT(11) | DOUBLE | DOUBLE | DOUBLE | DOUBLE | DOUBLE | DOUBLE |

Finally, the descriptive statistics are obtained by calling the (static) CalculateFinalSatistics method, which calculates the mean, unbiased variance, sample skewness, sample kurtosis, and G2 kurtosis estimators (min and max do not need to be updated). These final results are those that are ultimately presented to the user by the Baron (cf. § 6.7).

### 5.2.2 Correlative Haruspex-Specific Output Tables

Similarly, in learn mode, each correlative haruspex logs its results into the table **HaruspexCorrelativeSubResults** with primary key (RequestId, SequenceId, Rank):

| RequestId | SequenceId | Rank | SampleSize | Sum1A | Sum1B | Sum2A | Sum2B | SumAB |
|---|---|---|---|---|---|---|---|---|
| INT(11) | INT(11) | INT(11) | INT(11) | DOUBLE | DOUBLE | DOUBLE | DOUBLE | DOUBLE |

Subsequently, a MySQL trigger updates the summary table **HaruspexCorrelativeResults** with primary key (RequestId, SequenceId):

| RequestId | SequenceId | SampleSize | Sum1A | Sum1B | Sum2A | Sum2B | SumAB |
|---|---|---|---|---|---|---|---|
| INT(11) | INT(11) | INT(11) | DOUBLE | DOUBLE | DOUBLE | DOUBLE | DOUBLE |

Finally, the correlative statistics are obtained by calling the (static) CalculateFinalSatistics method, which calculates the means, unbiased variances, and unbiased covariance estimators, along with the linear correlation coefficient and the 4 linear regression coefficients. These final results are those that are ultimately presented to the user by the Baron (cf. § 6.7).

### 5.2.3 Multi-Correlative Haruspex-Specific Output Tables

In learn mode, each multi-correlative haruspex logs its results into the table **HaruspexMultiCorrelativeSubResults** with primary key (RequestId, SequenceId, Rank, MetricA, MetricB):

| RequestId | SequenceId | Rank | MetricA | MetricB | SumAB |
|---|---|---|---|---|---|
| INT(11) | INT(11) | INT(11) | INT(11) | INT(11) | DOUBLE |

Subsequently, a MySQL trigger updates the summary table **HaruspexMultiCorrelativeResults** with primary key (RequestId, SequenceId, MetricA, MetricB):

| RequestId | SequenceId | MetricA | MetricB | SumAB |
|---|---|---|---|---|
| INT(11) | INT(11) | INT(11) | INT(11) | DOUBLE |
| | | $-1$ | $-1$ | size |
| | | $i = 0, \ldots, m-1$ | $-1$ | first moments |
| | | $i = 0, \ldots, m-1$ | $j = 0, \ldots, i$ | second moments |

Here $m$ is the number of metrics; the number of rows in **HaruspexMultiCorrelativeResults** for each multi-correlative haruspex run is $1 + m + m(m+1)/2$. Finally, the multi-correlative statistics are obtained by calling the (static) CalculateFinalSatistics method, which calculates the means and Cholesky values. These final results are those that are ultimately presented to the user by the Baron (cf. § 6.7).

### 5.2.4 Common Haruspex Output Tables

In addition to the haruspex-specific output tables, in monitor mode all haruspices log reportable events into the table **HaruspexRequestsReportedEvents** with primary key ReportedEventId (auto-incremented, may not be sequential for a given RequestId):

| ReportedEventId | RequestId | Cause | Value |
|---|---|---|---|
| INT(11) | INT(32) | INT(32) | DOUBLE |

where RequestId, Cause, and Value indicate, respectively, the haruspex that reported the event, the type of event, and a numeric characterization of the event (whose meaning may depend on Cause and on the haruspex type).

To make it possible to locate the datum that generated a reportable event, table identifier and key in this table are logged for each event into the table **HaruspexRequestsReportedTableKeys** with primary key ReportedEventId:

| ReportedEventId | TableId | TableKey |
|---|---|---|
| INT(32) | INT(32) | INT(32) |

Note that CompId is not present in the reported event tables, because it is implicit in the combination (TableId, TableKey) in HaruspexRequestsReportedTableKeys. In fact, if and when correlations across multiple components are supported, there may be no single CompId associated with an event (this would of course involve changes to HaruspexRequestsComponents).

## 5.3 Example: Multi-Correlative Haruspex

We now illustrate the use of OVIS 2.0 haruspices with an application of the multi-correlative haruspex to resource characterization. In this approach, anomalous behaviors are sought by

1. calculating (with "training data") or devising (with "expert knowledge") mean vectors and covariance matrices – and thus, implicitly, a multiple linear regression model – for a set of tuples of variables of interest, and

2. examining how individual observations of these tuples of variables of interest deviate from the aforementioned model; such deviations are characterized in terms of the significance level to which they correspond when the mean vector and covariance matrix are made those of a multivariate Gaussian model. Note that this is directly related to the multivariate Mahalanobis distance computed with the mean vector and covariance matrix.

Figure 11: Actual rendering of the Red Storm platform zoomed in on the partition on which data were taken. The nodes are colored red if below the user-defined probabilistic threshold for being too unreliable and green otherwise. Grey indicates there was no data in the display time window for that resource for the metric being displayed.

For instance, Figure 11 displays a simple use case where only one pair of variables is of interest to the analyst, namely PROCPIC_0_CORE and PROCPIC_0_Proc_Int, which we will respectively denote $A$ and $B$.

When the first phase of the process described above is meant to be calculated (as opposed to devised, e.g., using expert knowledge), then the "Learn" mode of the haruspex is turned on prior to the execution of the haruspex on a set of training data. This is the case in the example of Figure 11: specifically, all observations $(a,b)$ of $(A,B)$ between the specified start and end times (respectively 10:52:02 a.m. and 4:45:02 p.m. on November 8, 2007) on all components called `rsnode`$n$, where $n$ varies between 1 and 3000, are used to "Learn" a model. To ensure that the number of observations are the same for each component, the data is interpolated by taking the most recent observed value at even time intervals for each component.

As a result, a mean vector and a covariance matrix are calculated, and are available to the user in the "Learn" tab (not selected in the figure).

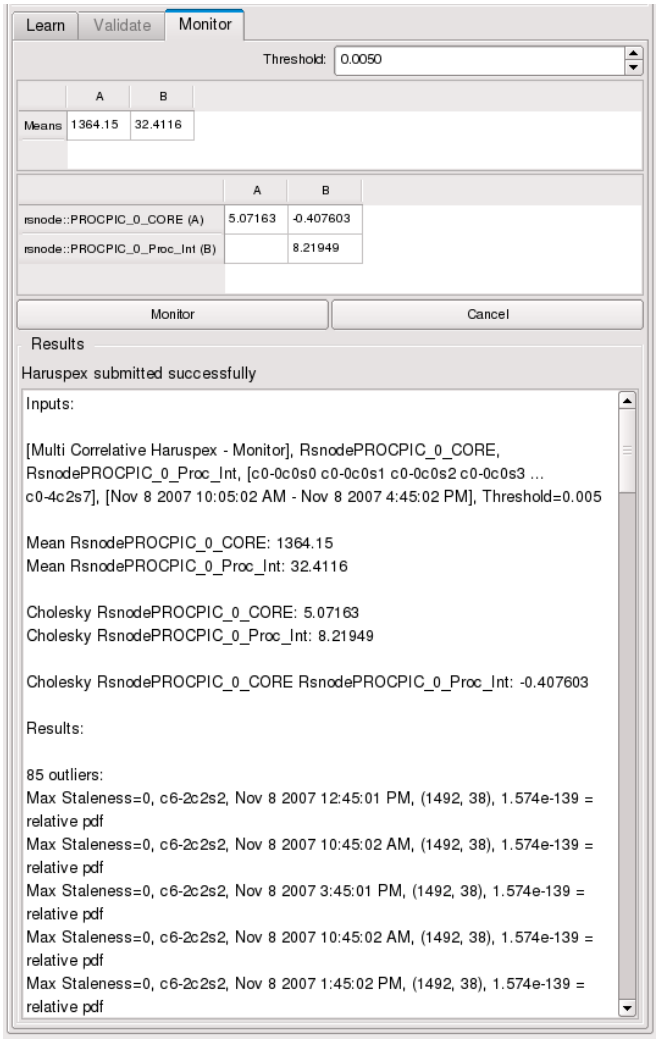



Figure 12: The interface in Figure 11, zoomed in on the analysis output.

Note that the second phase of the analysis process, called "Monitor", can be performed on either the same data set used to infer a model, or on a different data set. For simplicity, the former option is the case in our running example. Therefore, as illustrated again in Figure 11 and Figure 12, under the "Monitor" tab, one can see the mean vector and Cholesky-decomposed covariance matrix that have been calculated by the haruspex during the "Learn" phase. In particular, the means $\mu_A = 1364.15$ and $\mu_B = 32.4116$ as well as the covariance matrix

$$\Sigma := \mathrm{cov}(A,B) = U^t U,$$

where

$$U = \begin{pmatrix} 5.07163 & -0.407603 \\ 0 & 8.21949 \end{pmatrix},$$

are those of the underlying (bivariate, in this case) linear regression model.

It is beyond the scope of this article to delve into too many details about multiple linear regression models and their relationships to multivariate Gaussian distributions; one only has to know that the underlying linear model is mapped into an $N$-variate (bivariate in our running example) Gaussian model, whose probability density function (PDF) is, by definition,

$$f_X(x) := \frac{1}{(2\pi)^{N/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^t\Sigma^{-1}(x-\mu)\right),$$

where $x^t := (x_1,\ldots,x_N)$ is the observation of an $N$-tuple of interest. In our bivariate example, this simplifies into

$$f_{(A,B)}(x) = \frac{1}{2\pi|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^t\Sigma^{-1}(x-\mu)\right),$$

where $x^t := (a,b)$ and thus $(x-\mu)^t = (a-\mu_a, b-\mu_b)$. (Note that the inverse covariance matrix $\Sigma^{-1}$ is computed only once, by means of the Cholesky decomposition.) The argument of the exponential is $-\frac{1}{2}$ times the squared Mahalanobis distance, which is the natural metric associated with the multivariate distribution. The significance level of observation $x$ is defined as the probability (in the Gaussian model) of observing a Mahalanobis distance greater than that of $x$. This significance level is a natural choice of cumulative distribution function (CDF) for the multivariate Gaussian distribution; it ranges from 1 for a central (mean) observation to 0 for observations infinitely far from the mean vector.

With this in mind, we define an outlier as any observation $x$ of $X$ whose significance level is less than a user-specified threshold $\tau$ (typically $\tau \ll 1$). If observations accurately follow the multivariate Gaussian model, then a fraction $\tau$ of observations should meet this criterion. Observations may not follow the inferred model, however, either because the data are non-Gaussian or because the data being monitored have a different distribution from the training data. Nevertheless, the computed significance level is useful in assessing the deviance of an observation.

A simpler description of the significance level is possible in the bivariate case. There, the significance level happens to equal the exponential factor in the Gaussian PDF. This factor can then be described as the relative probability (normalized to the maximum of the PDF), and an outlier can alternatively be defined as an observation $x$ with

$$\frac{f_X(x)}{\max_{\mathbf{R}^2} f_X} < \tau.$$

As shown in the "Monitor" tab of Figure 12, a threshold value of $\tau = 0.005$ was chosen, resulting in 85 outliers being reported by the haruspex, and listed in the lower right text window of the user interface. For example, the first of these outliers corresponds to an observed value of $(1492, 38)$, which, in the context of the underlying model, has a significance level (or relative probability) of $\approx 1.574 \cdot 10^{-139} < \tau = 0.005$, making it an outlier according to our definition. (Such a vanishingly small value indicates that the data are non-Gaussian, since an event with actual probability of order $10^{-139}$ would not realistically occur.) In turn, in the cluster view of Figure 11, all components

evincing outlier behavior at the time shown in the view are colored in red, whereas other components appear in green (data were not collected on the grayed-out components during the time interval of interest).

# 6 Baron

The Baron is the graphical interface of OVIS which enables exploration of the data. The Baron allows the user to:

1. select a cluster and a relevant database
2. visualize cluster geometry in a physically accurate display
3. visually inspect raw variable values and those variables relative to model calculations on the physical display
4. create haruspices and inspect the results of their analyses, both textually and graphically
5. view historical data, browsing through time history both manually and animatedly
6. view live data, updating in real-time, and
7. tune many display parameters.

Features and capability of the Baron are described in this section. Figure 13 is a picture of the Baron with the elements referred to in this section labeled.

## 6.1 Cluster and Database Selection

The Bookmark Editor allows you to specify to which database you want to connect. Figure 14 (left) shows selection options filled out for connecting to local database holding the test data set for the Whitney cluster. Use of this data set is discussed in §7.1. In addition to selecting a database, you may enter connection parameters (user name and, possibly, a password) in the Server Connection window in shown in Figure 14 (right). **WARNING: In the ServerConnection window, if you choose the option to remember the password for any of the different OVIS_⟨clustername⟩ databases, the user name and password on your database will be stored in plain text in the file** `${HOME}/.config/Sandia/ovis.conf`.

The yellow star icon indicates a user-specified static entry. The blue globe icon indicates an automatically-populated entry, representing an available shepherd as advertised by Avahi.

## 6.2 Adding Views

Any number of physical and analysis panes may be shown simultaneously. You can create additional panes by either clicking on the New Pane Icons in the upper left corner of Figure 13 or the Split Pane buttons in the upper right corner of any existent pane, also seen in Figure 13 which split

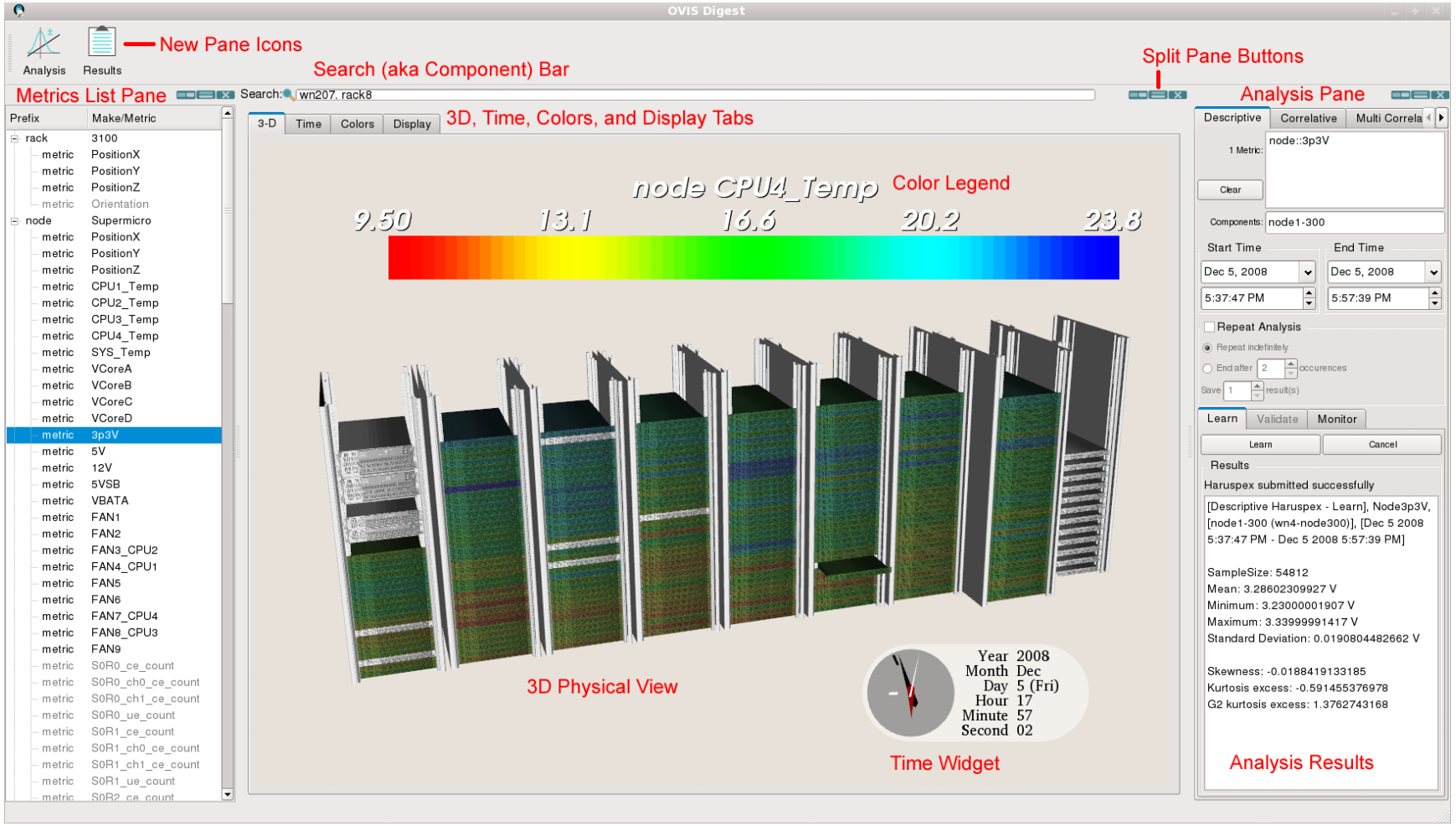



Figure 13: Overview of the elements of the Baron with annotation.

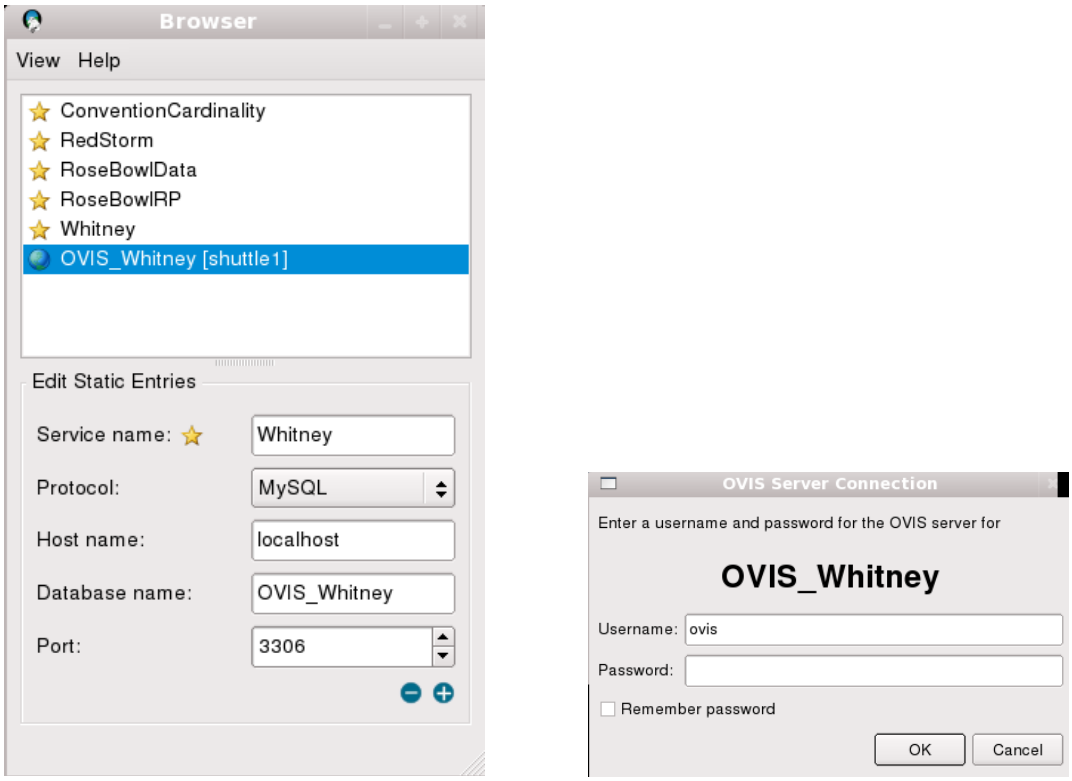



Figure 14: Bookmark Editor (left) and Server Connection (right) windows.

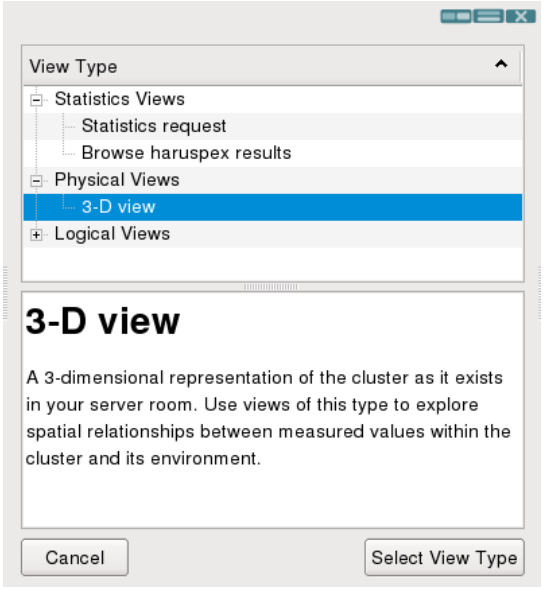



Figure 15: Options for instantiating a new pane.

the existent pane in the direction indicated in the buttons. The buttons include one by which a pane can be closed.

When a split is selected, options for the new pane are presented and can be selected as in Figure 15.

## 6.3 Rotating, Panning, and Zooming the 3D View

The middle pane in Figure 13 is a 3D interactive physical representation of the cluster of interest. Moving the mouse while pressing the left mouse button will rotate the 3D view. Moving the mouse while pressing the middle mouse button will pan the 3D view. Moving the mouse while pressing the right button will zoom the 3D view. Also, rotating the mouse wheel button will zoom the 3D view. Selecting the *Display* tab (shown in Figure 13) presents the user with the option to reset the physical view to its original position.

## 6.4 Metric Drop

On the far left of Figure 13 is a pane which lists the various component types in this data set and the metrics which are taken upon them. Metrics can be selected with the mouse and dragged and dropped upon the physical view (Middle pane). The physical view will then color the elements by the selected metric. A color bar will indicate the color-value mapping. By default, the range has as its max and min values the max and min values for that metric for the time displayed (More on Time in §6.8). The range can be overridden as described in §6.5.

Components having no value at that time (or within the relevant fade period, described in §6.8) will be shown as gray.

Drops of analyses onto the physical view are also enabled and are discussed in §6.7.

## 6.5 Setting the colors

The Baron provides a default background color, as well as default color scheme, scale and ranges for the component variables. These can be modified through the *Colors* tab – the tab is shown in 13), with the content of the tab illustrated in Figure 16. The Components section of the Color tab allows one to override the default range for the color legend for metrics. This can be done by selecting `Override default range` and specifying the range explicitly. If this is done, the color range for the metrics will be fixed, even as one animates through time (see §6.8).

Setting the range manually allows easy visual comparison of the distribution and locations of regions of interest across time and across data sets. Selecting a subset of the overall possible range allows one to see finer grained detail within a section of interest. For example, if the range for a

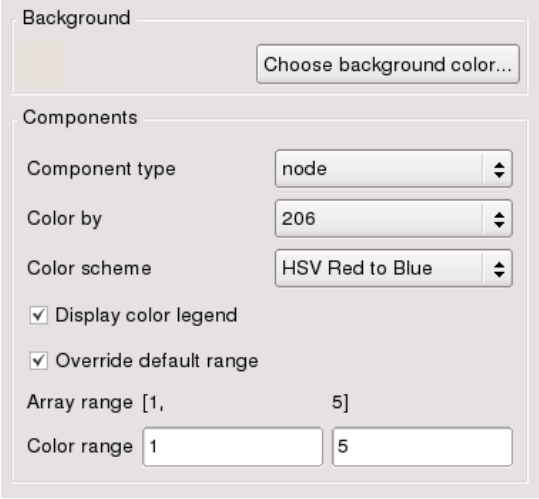



Figure 16: The color tab, where the color legend can be set.

particular metric ranges from 0-100, one may be primarily interested in details of the values in the upper end of the range, and therefore may set the color range from 80-100, for instance.

All color related options in this section will be retained as described in §6.10.

## 6.6 Search bar

The Search Bar (aka Component Bar) 13 is used to pop out components in the physical display. This is shown in Figure 13 where we have popped out a node and a rack. Components to pop out can be specified by short name (e.g., wn207) or by component type and number (e.g., rack8). The advantage of the former is that the short name is generally well known. The advantage of the latter is range notation is supported (e.g., "node10-14,node57-65").

Some of the analysis monitor results can also be dropped onto the Search Bar, popping out the outlier components. This is described in more detail in §6.7.

## 6.7 Haruspices

The Baron provides a graphical interface for creating and obtaining the results of the haruspices (cf. 5).

A typical analysis pane is shown in Figure 17. It contains regions in which to input the metric or metrics of interest, the time range of the calculation, and the components involved in the calculation. The metrics can be populated by dragging and dropping them from the metric list. The components can be specified by component type and number, in which case ranges are supported, or by short name.

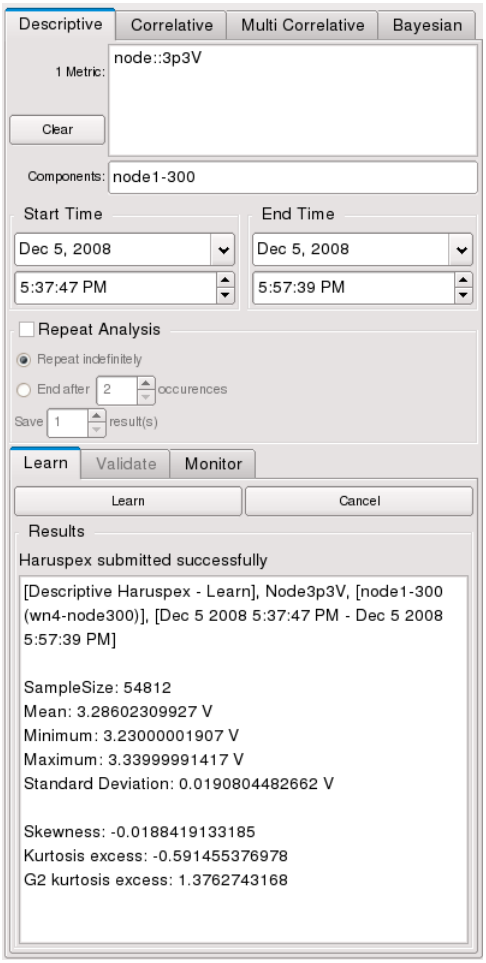



Figure 17: Descriptive learn Analysis pane where the metric, components, and time range for analysis are specified

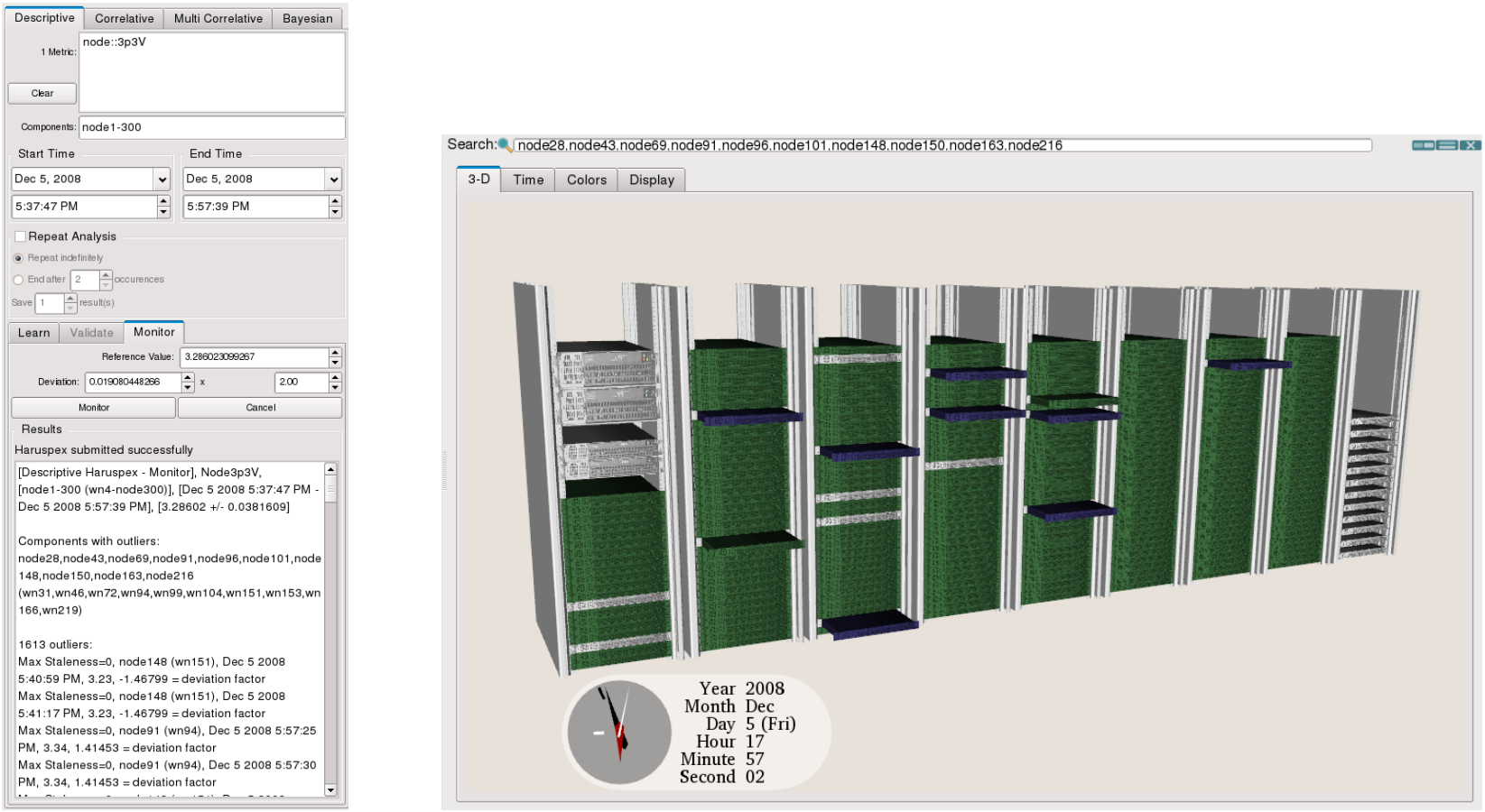



Figure 18: Descriptive monitor Analysis pane (left) and associated Model drop.

learn analyses learn a model; monitor analyses determine outliers given a model. For more information on the learn and monitor phases of OVIS analyses, please refer to §5.1.

After a learn analysis, one can click on the Monitor tab and the Monitor window will have the the model parameters automatically filled with the results of the learn analysis. In the case of the Descriptive Learn analysis in Figure 17 the associated Descriptive Monitor analysis is shown in Figure 18 (left). The Mean and Deviation are automatically populated from the learn; the number of deviations is by default 2. All of these values can be changed by the user. In the case of the Multicorrelative analysis, the parameters that are automatically filled are those of the matrix described in §5.3.

Clicking the Learn or Monitor buttons on the pane start the analysis. When the results in an analysis window are not current, either because new parameters are being entered into an analysis pane or because the analysis has not yet returned, the result area of the window is colored pink.

After a monitor analysis, outliers are displayed in the results window. These outliers can also be displayed in the physical display, by dragging and dropping the analysis onto the physical display (Grab where it says "Haruspex submitted successfully"). Supported drops are

- Descriptive Monitor - colors everything below the threshold red, above blue, and in between green. See Figure 18 (right).
- MultiCorrelative Learn - colors on a scale from red to blue everything by its significance level. See Figure 28 (top) in §7.1.
- MultiCorrelative Monitor - colors red everything with significance level below the threshold, green everything above the level. See Figure 28 (bottom) in §7.1.

In the first and third cases, the values corresponding to the color bar legend are meaningless - they only serve to make "good" values green and bad values "red" or "red/blue". The multicorrelative cases are described in more detail in §7.1.

Outliers will pop out of the physical display if the analysis is also dragged to the Search Bar. So that you may be sure that you have successfully dropped the results properly, the Search Bar will flash green a few times after the drop.

The *Repeat Analysis* button is currently not enabled. When enabled it will automatically recalculate the analysis including any newly collected data. This will ensure that the model is current and will allow the user to note model changes with time. This feature will be enabled in a future release.

## 6.8 Time Features

One can manually scroll through time or have the Baron automatically animate playing through time, and view the current state in the physical display.


Figure 19: The user interactive time widget allows the user to scroll through time in the physical display.

The user interactive time widget 19, shows the current time in the physical display. There are marks/hands for month, day, hours, minutes, and seconds that can be grabbed and pulled forward and backward in time with the current state shown in text as well.

Additional handling of time is done in the *Time* tab – the tab is shown in 13), with the content of the tab illustrated in Figure 20.

The first check-box controls the visibility of the time widget. The box below enables animated playing through time. Setting the `x RealTime` entry to values greater than zero enables automatic playback at the specified rate, where 1 equals real time. Setting the `Frame Rate (target)` entry specifies the maximum number of times per second that the baron should update and redraw the cluster. High values require more communication with the database but will make color fades and the motion of the clock hands on the time widget appear smoother. Low values require less communication but may result in a "jumpy" looking interface. It is possible to specify a frame rate that the baron is unable to produce. In this case, it will redraw the cluster as quickly as it can.

After the `x RealTime` and `Frame Rate (target)` entries have been set, the user can press the space bar (on the keyboard) in the 3D View tab to start and stop the clock. Note that stopping the clock will only freeze the display, and the internal clock will continue to progress; the 3D View will reflect the data at the frozen time.

The playback time section allows one to manually set the initial time of the clock. The `Earliest` button will set the time to the earliest time in the database; the `Now` button will set the time to the current time on the machine. After adjusting the time, click the `Apply` button for the changes to take place.

Data is recorded in the database at the fidelity of a second. Because of issues such as clock skew, database insert times, etc., it is desirable to see all data not at a given point in time, but within a window of time. For this region a `Fade Period` can be set that will allow components in the physical data to be colored by the data value corresponding for that time nearest in time to that shown on the clock within the specified fade period. For example, for the Whitney database, described in §7.1 data is taken on 5 second intervals, so one should set the Fade Period to 9, and preferably greater, so that all components will be colored by a timely value. In order to distinguish the age of values, the component color will also fade out as the data value gets increasingly distant from the clock time. There is no correct value of the fade period – this is determined by the

frequency of data collection/inserts and the user's desire regarding the fading effect. For example, if the user finds the fading distracting, a longer time may be desirable; if the user is specifically trying to investigate when components cease to report a smaller fade time is more appropriate. Note also that longer fade periods require more data to be sifted through so that longer fade periods result in decreasing performance.

Note that currently the fade period is a global variable pertaining to all physical views in the Baron.

All time related options in this section will be retained as described in §6.10.

## 6.9 Haruspex Requests View

The Haruspex Requests view 21 can be generated via the New Pane icons in 13. This is a table that lists the previously requested analyses, displaying the RequestId and some parameters of the request as described in §5.2. Single clicking an entry brings up the results. Double clicking an entry instantiates the corresponding Analysis view, filled out with the request and results. This allows the user to view the results without having to redo the analysis. The analysis pane generated in this way is similar to other generated panes and can be dropped on the the physical display and Search Bar, as usual.

## 6.10 Saving State

State, including color bar ranges, time ranges for analyses, time shown in the physical pane, etc, in plain text in a file in `$HOME$/.config/Sandia/ovis.conf`.

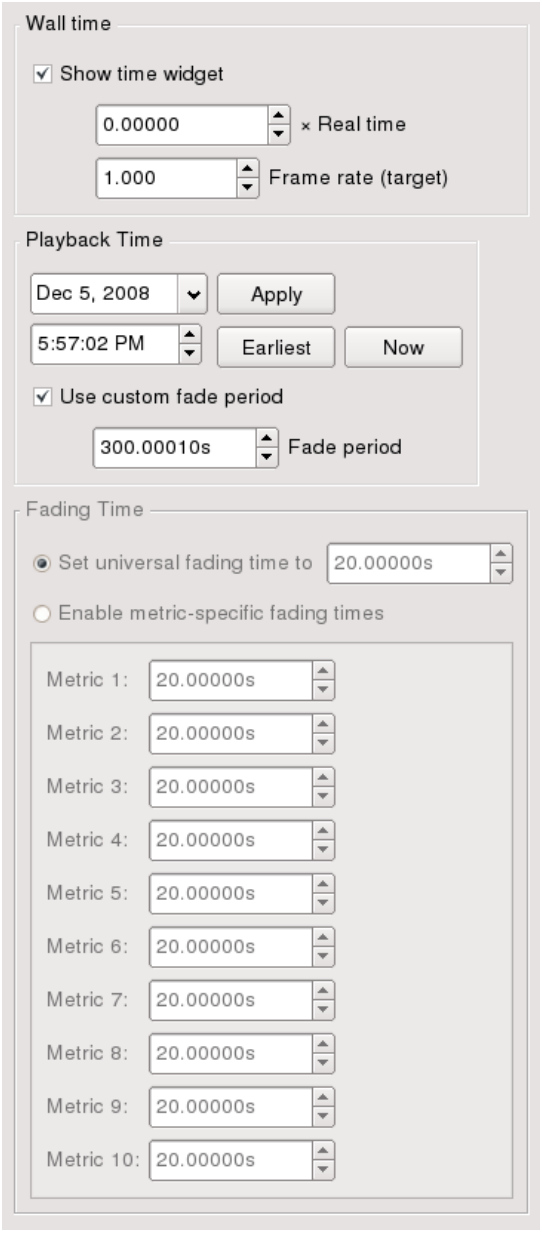



Figure 20: The Time tab, which allows the user to set the time; choose to play through time; and set the fade period.

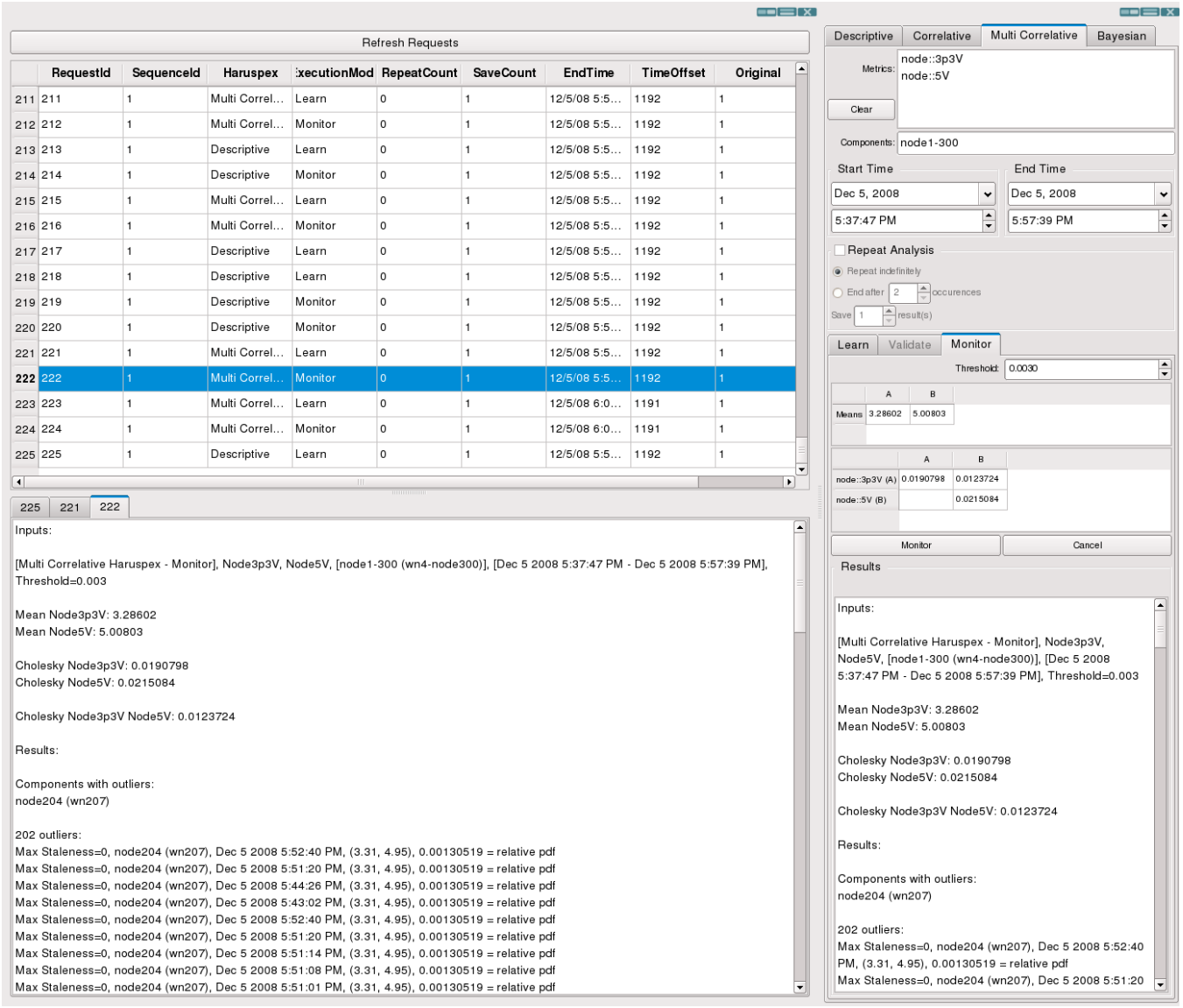



Figure 21: The Baron Requests view which allows one to examine previous analyses. The upper left is an interactive table for selecting previous analyses. Single clicking an entry brings up the results in the lower left. Double clicking an entry instantiates the corresponding Analysis view.

# 7 Examples

The OVIS 2.0 release tar ball comes with three example cases. The first includes a set-up file and example data from a cluster. The example data is released as a separate tarball. The second and thrid are a set ups file, that with minor modifications will take and display data from your local machine.

See §2 for info on the mysql settings and general system settings before beginning.

## 7.1 Whitney Example Data

This case involves analysis of some test data already gathered from a cluster. It illustrates use of the Baron in case where we find anomalous behavior when we consider the behavior of two variables taken in conjunction (Multicorrelative analysis) that would be insufficiently captured, if one were to independently consider the behaviors of the two variables.

Data is not gathered, but rather is loaded from a mysqldump of previously gathered data. Relevant files are 1) `whitneydemo.ovdb` in the OVIS data directory which is the set-up data file which was used in the actual data collection and established the cluster display arrangement and 2) `mysqldump.OVIS_WhitneyRelease.sql.tgz` which is the mysqldump of the database and exists in a separate release data tarball.

We illustrate the analysis by building the display in Figure 22 which consists of three analyses, two Descriptive and one Multicorrelative using the variables in the two Descriptive analyses. Further, there are physical displays associated with each analysis, where the displays easily illustrate outlier behaviors relative to the resultant analysis models.

### 7.1.1 Getting Started

1. First load the mysqldump. This requires creating the empty database in mysql and then decompressing and loading the data file.

   ```
   mysql> create database OVIS_Whitney;
   mysql -u ovis OVIS_Whitney < mysqldump.OVIS_WhitneyRelease.sql
   ```

2. If this is your first use of the MySql database with OVIS it is recommended that you run the database effector to load the user defined function for initiating analyses as described in §4.2.

3. Edit the **StartupData** table so that OVIS will recognize the shepherd on your machine for performing analyses:

   ```
   mysql> use OVIS_Whitney;
   mysql> select * from ComponentTypes;
   ```

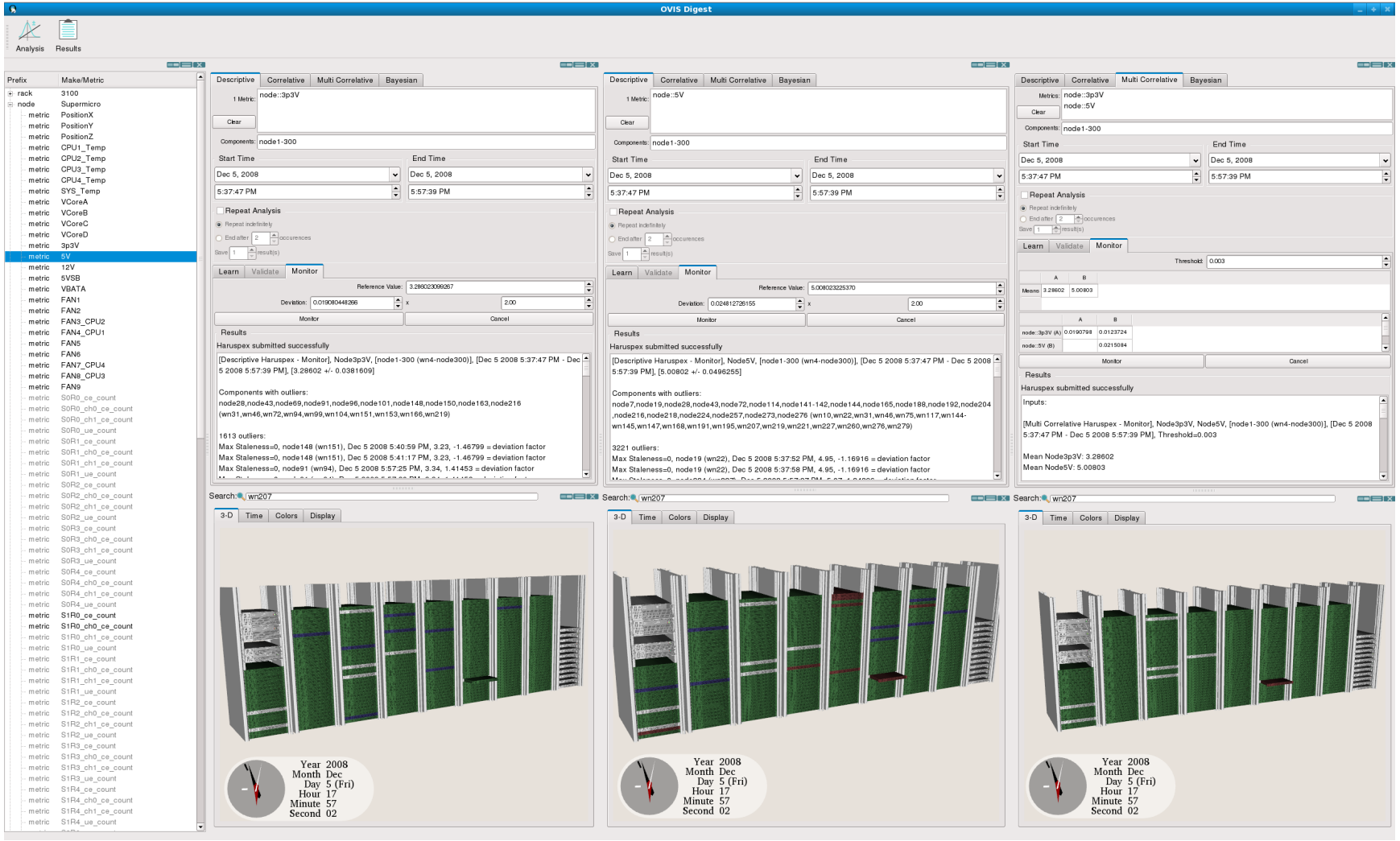



Figure 22: Using the Baron for analysis. Three sets of analysis and model drops onto the physical display are shown. Two are descriptive statistics and one is a multicorrelative analysis. Outliers relative to the relevant models are colored in the figures.

(this will show you that the shepherd has CompType 5)

```
mysql> select * from ComponentTable where CompType=5;
```

(this will show you that the shepherd has CompId 314)

```
mysql> select * from StartupData where CompId=314;
```

(this will show you the current allowable shepherds)

```
mysql> update StartupData set AddressData="XXX" where
     CompId=314 and AddressType=2;
```

(replace XXX with the hex version of your IP address)

```
mysql> update StartupData set AddressData="XXX" where
     CompId=314 and AddressType=0;
```

(replace XXX with the hex version of your MAC address)

4. Start the shepherd:

```
cd /path/to/ovisBuildDir
./bin/shepherd --name=Whitney \
     --database=mysql://ovis@localhost/OVIS_Whitney
```

If you get a warning at this point like

```
Shepherd was unable to determine its own component ID.
   This will cause haruspex calculations to be unreliable
   at a minimum. Set a component ID in the StartupData table.
```

then you have not edited the **StartupData** table properly.

5. Start the baron:

in a different window but the same directory run:

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/path/to/ovisBuildDir/lib
./bin/baron
```

6. Choose the Whitney database in the Baron (The Bookmark Editor and ServerConnection windows are described in §6.1):

   (a) When the Browser Window comes up select: *View→Show Bookmark Editor*

   (b) Fill in the following fields:

   - Service name = Whitney
   - Protocol = MySQL
   - Hostname = localhost
   - Database name = OVIS_Whitney

   (c) Click on the plus sign

   (d) Select "Whitney" in window

(e) When the OVIS Server Connection window comes up you should see ”ovis” as the user name and nothing for the password. You can use the default, if you have enabled user ovis the appropriate permissions on the OVIS_Whitney database; otherwise you can use the user name and password of your choice. Note that these will be stored (as described in §2) in plain text on your machine. Click ”OK”.

(f) At this point the OVIS_Digest (baron window) should appear with the cluster displayed.

7. Set the Fade Period in the Time Tab to 30 seconds. (The Fade Period and Time tab are discussed in §6.8. Determination of the correct value based on the innate frequency of the data collection (in this case 5 second intervals) and the preference of the user as to the fading effect. If this is the first time running the Baron, the date/time may be set to January 1, 1970, GMT. To change the date/time to the earliest time at which there is data for this cluster, click on the “Earliest” button in the Time Tab of the 3D View.

### 7.1.2 Using the Baron: Displaying Raw Metric Values

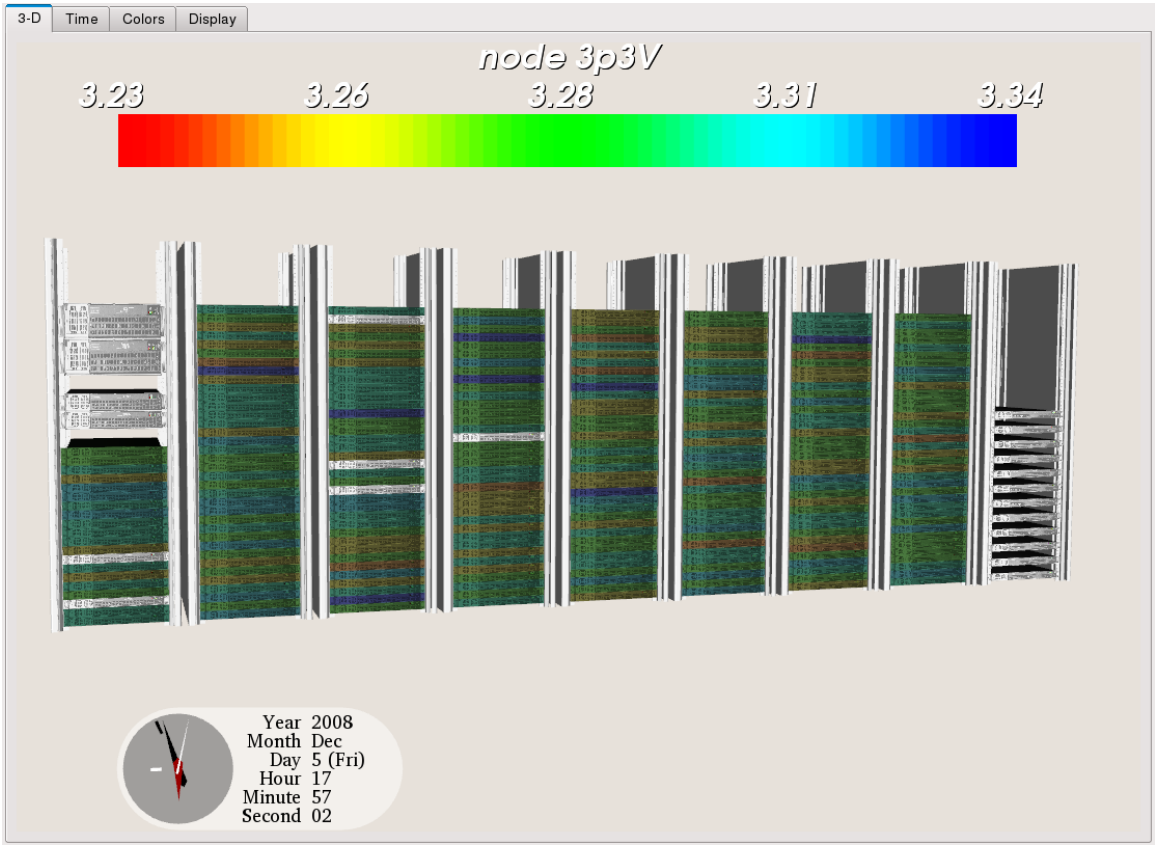



Figure 23: Raw metric values on the physical display pane.

To display raw metric values in the physical display:

1. Click on “node” in the left hand menu to see the available metrics for display for the node.

2. Drag and drop a metric onto the display, such as CPU1_Temp.

The nodes should become colored by value as shown in Figure 23. The range of the values in the color bar is determined by the min and max metric values exhibited at the time in the display, unless overridden by settings in the Color tab (described in §6.5).

### 7.1.3 Using the Baron: Performing Analyses and Displaying Model Comparisons in the Physical Display

With the analysis capabilities, you can build a model of data and determine outliers relative to that model, presented both textually and in the physical display. The analyses and model drops in this section are those in Figure 22.

Click on Analysis in the upper left of the window and a Descriptive Analysis pane should appear. The default time range is the entire time range in the database. Do the following:

1. Drag and drop the metric 3p3V onto the analysis metric window .
2. For the components entry, fill in "node1-300". Components can be specified by either their canonical names or their component type and number (as they are in this case). If they are specified by canonical names, note that ranges are not supported as canonical names can have hyphens within them.
3. Click on Learn

Analysis results should appear as in Figure 24 (left).

To determine outliers relative to the results of the Descriptive Analysis:

1. Click on the Monitor tab in the same pane. Note that the results of the learn analysis have populated the monitor options. By default, the outlier threshold has been chosen to be 2 standard deviations. You may change any of these values, but for this example keep them as is.
2. Click on the Monitor button

Analysis results should appear as in Figure 24 (right). In the result window will be a list of components which satisfy the outlier criteria at any time during the time range as well as at least a partial list of details of the outliers, including component, time, and metric value.

After an analysis, outliers can also be displayed in the physical display, by dragging and dropping the analysis onto the physical display (Grab where it says "Haruspex submitted successfully"). Supported drops are

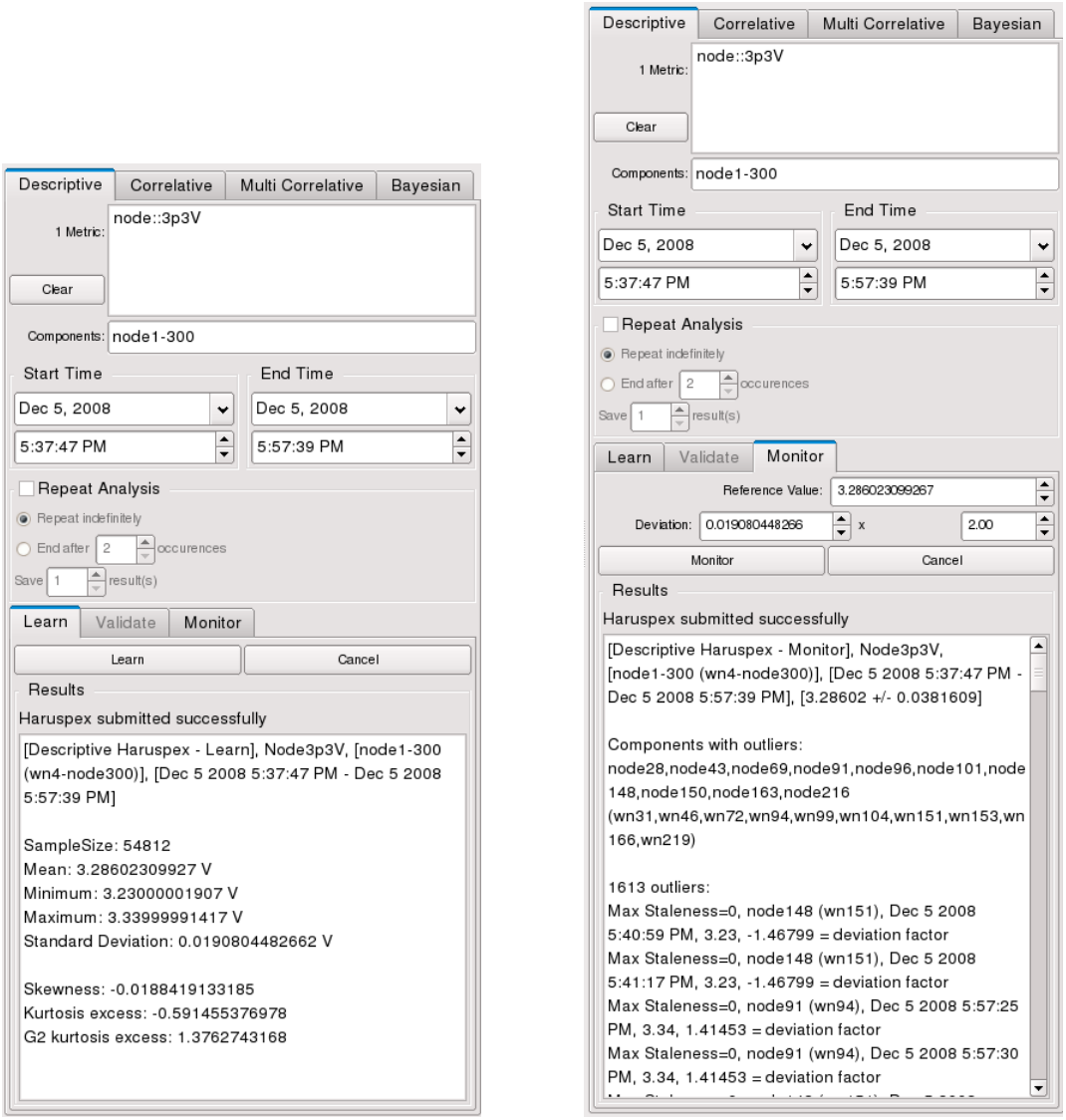



Figure 24: Descriptive learn (left) and monitor (right) Analyses panes.

- Descriptive Monitor - colors everything below the threshold red, above blue, and in between green.
- MultiCorrelative Learn - colors on a scale from red to blue everything by its significance level.
- MultiCorrelative Monitor - colors red everything with significance level below the threshold, green everything above the level.

Additionally, outliers will pop out of the physical display if the analysis is also dragged to the component list window.

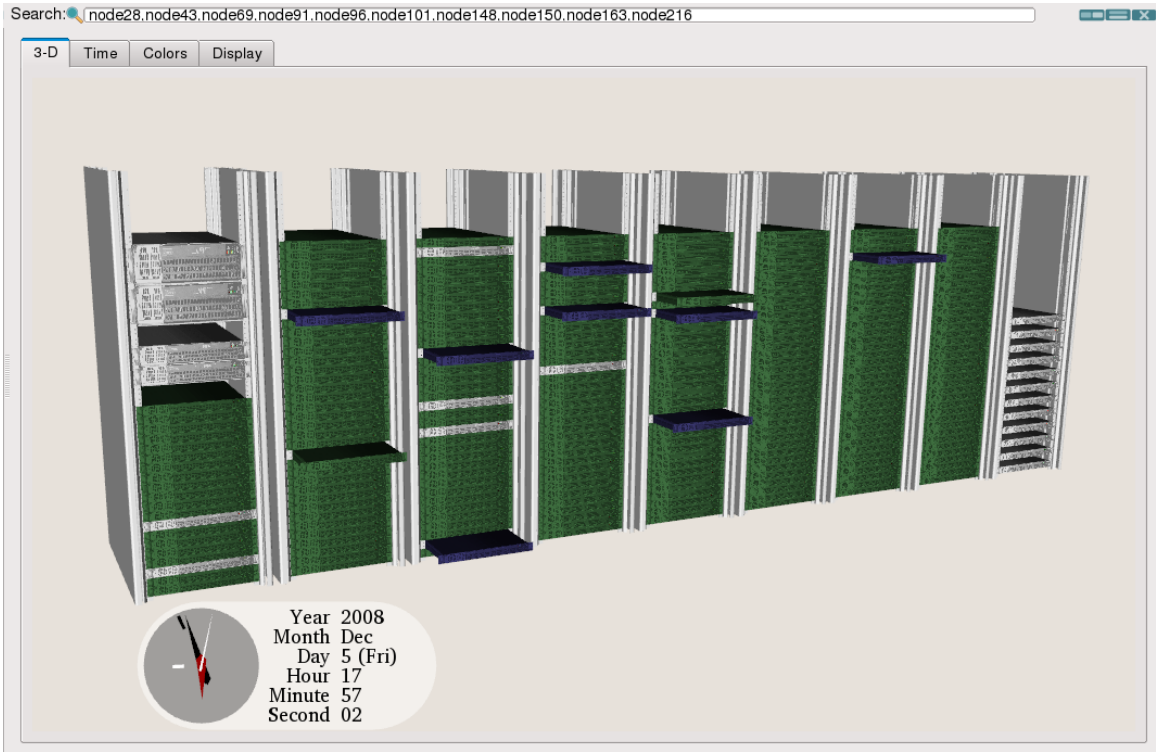



Figure 25: Descriptive monitor Model drop

This first is shown in Figure 25. Note that depending on the time chosen, components may pop out that are not colored as outliers at that time, as outliers at any one time may not be outliers at all times. The Multicorrelative drops will be illustrated below after the Multicorrelative analysis is illustrated. (The analysis and model drop are the leftmost column in Figure 22).

Splitting the pane (see Figure 15 in §6.2) and performing a similar descriptive analysis for 5V proceeds similarly, with the results shown in the middle analysis and physical display column in Figure 22. Note that for the 5V cases there are outliers both below (red) and above (blue) the mean.

Finally we illustrate a Multicorrelative analysis, using both the 3p3V and 5V metrics:

1. Drag both metrics to a new analysis pane

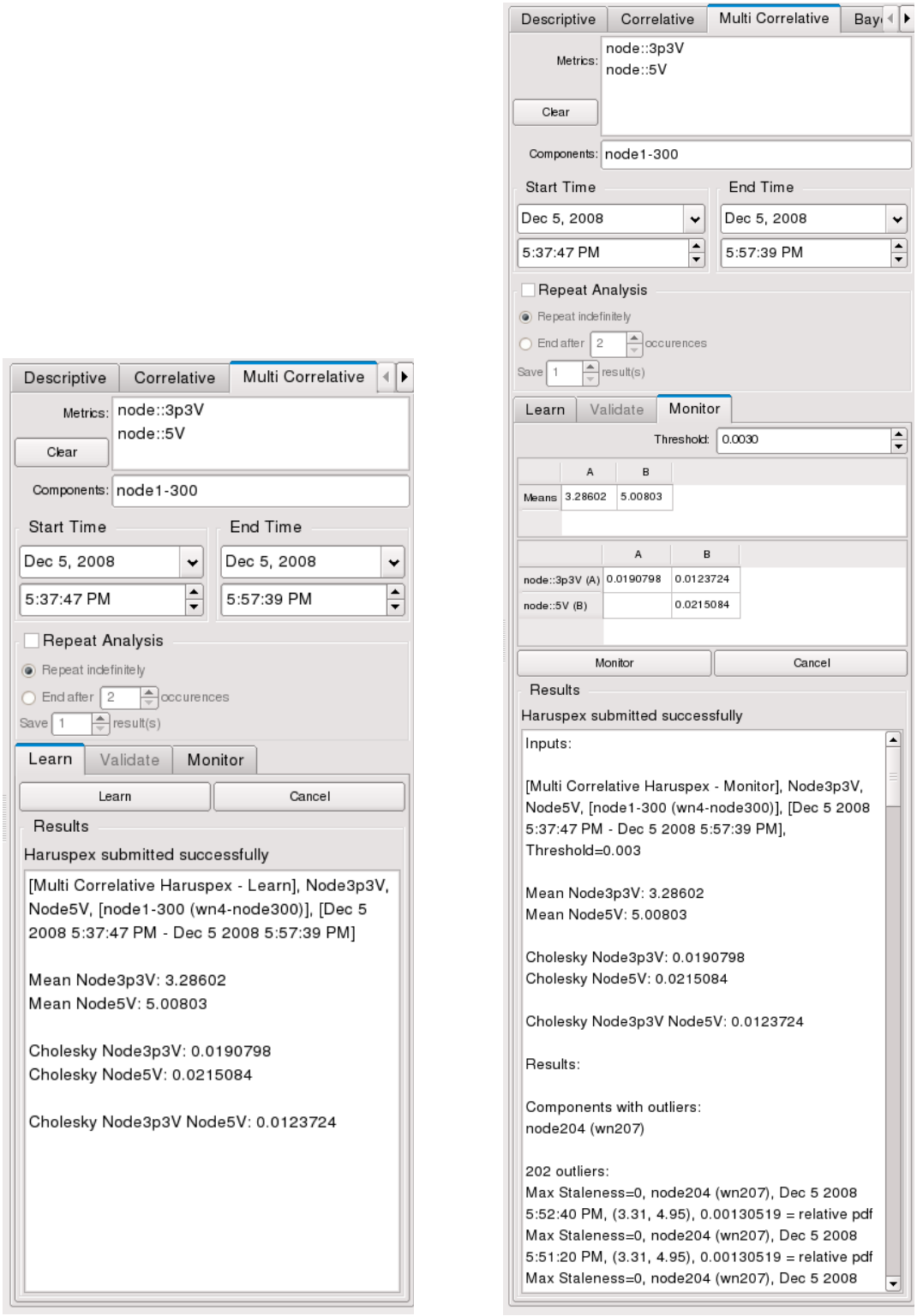



Figure 26: Multicorrelative learn (left) and monitor (right) Analyses panes.

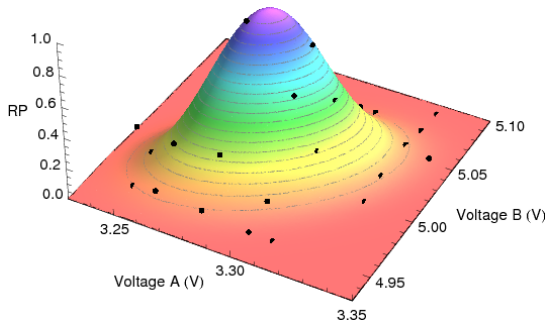



Figure 27: Evinced data compared to the calculated model for the Multicorrelative Analysis on the two metric previously studied as single metrics in the Descriptive Analyses.

2. Click on Learn

The resultant pane is shown in Figure 26 (left).

This analysis can then be dragged to the physical pane as shown in Figure 28 (top). In the physical pane model comparison, data for that time is compared to the model and colored according to significance level. Figure 27 shows the model surface calculated in this case with evinced data (over the entire time range) indicated on the plot. (This is not a figure that you can generate with OVIS via the Baron).

The Multicorrelative Monitor analysis determines probabilistic outliers given a user specified significance level:

1. Select the Monitor option on the previous Multicorrelative analysis pane. This populates the analysis with the model values recently calculated.
2. Change the probabilistic threshold to 0.003
3. Click on Monitor

Results are shown in Figure 26 on the right. Note that there is only one node exhibiting outlier behavior in the multicorrelative analysis, as compared to the two single variable analyses. The analysis can be dragged to the physical pane as shown in Figure 28 (bottom). (This analysis and model drop are the rightmost column in Figure 22). We have further dropped the analysis on the component selection which pops out the the node exhibiting outlier behavior in the multicorrelative analysis. It is easily seen in the physical view that the outlier node in the multicorrelative analysis is not an outlier in the 3p3V metric at the time shown and that outliers in one of the metrics do not necessarily exhibit outlier behaviors in the correlation analysis.

Analyses such as these can be used not only to determine outliers but to determine variable dependencies. This information, when combined with event data, such as, for example, node failure

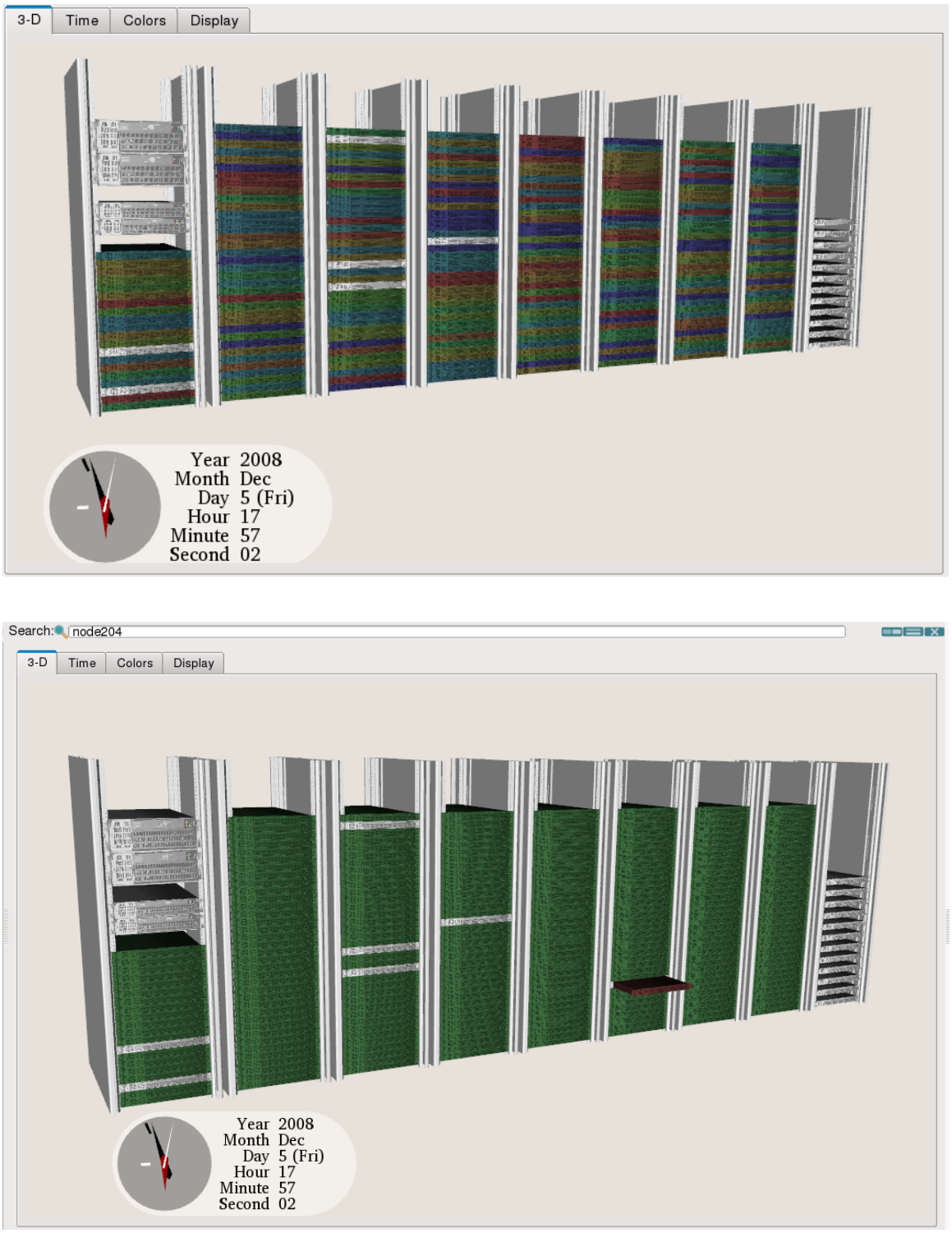



Figure 28: Multicorrelative Learn Analysis dropped onto the physical layout (top) and Monitor Analysis drop (bottom). Nodes are colored by significance level in the Learn drop. Nodes are colored green/red relative to the probabilistic threshold in the Monitor drop. Note that outliers in the Multicorrelative Analysis are not necessarily outliers in the single metric Descriptive Analysis and vice versa (only 1 outlier in the Monitor drop).

data, can be used to determine if outliers in a metric or a combination of metrics can be used as advanced indicators of the event of interest.

### 7.1.4 Using the Baron: Playing Through Time in the Physical Display

The Baron also supports playing data through time. Details can be found in §6.8. For this example, in the Time tab shown in Figure 20:

1. set the Playback time to be the initial time given in the analysis window
2. click on Show time widget
3. set 1xRealtime

In the physical display you should see the time animation of the data. Colors shown indicate the data value at the time currently shown on the clock and the clock should be changing with time.

## 7.2 Localhost Demo Files

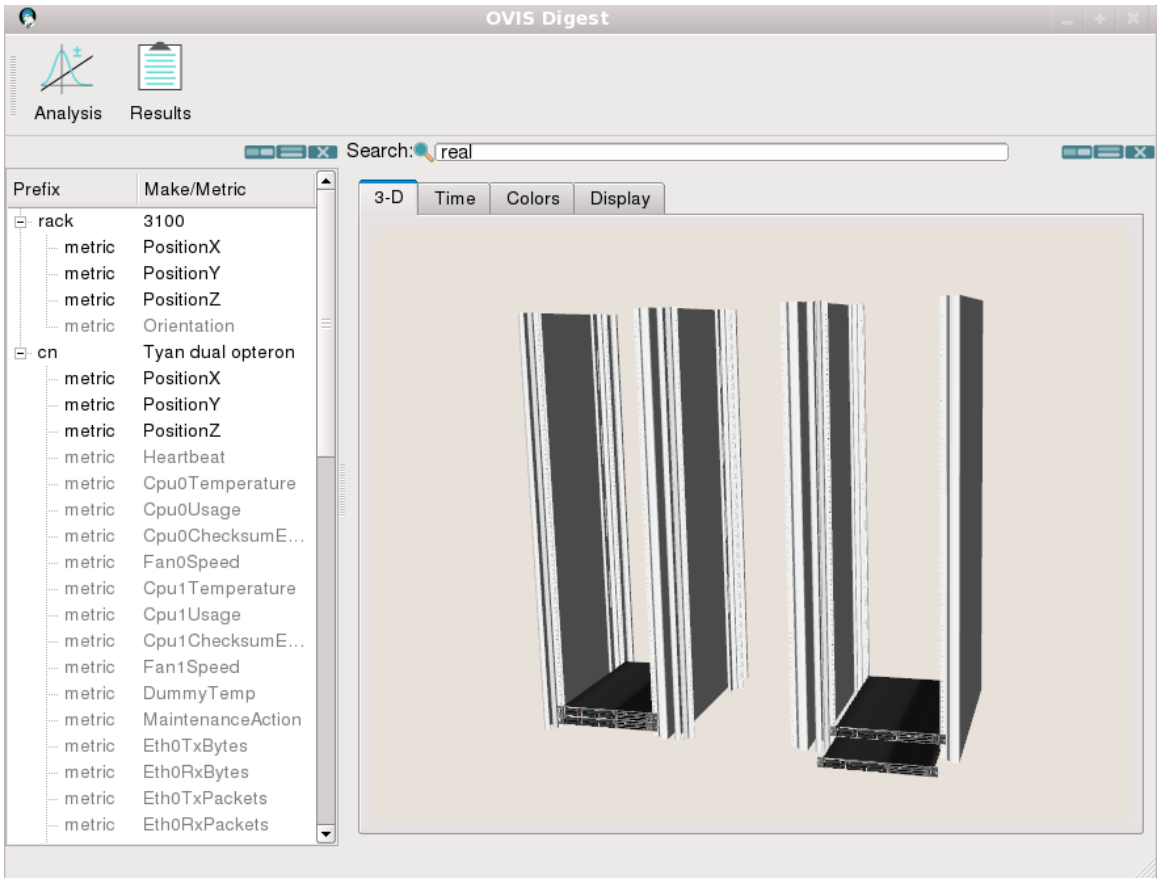
This section describes two example cases which can be used on your local machine. They further illustrate use of local and remote samplers for core utilization information, as described id §4.3.1.

### 7.2.1 Node level display

This case collects data from your local machine for analysis and display. In the OVIS source data directory is the file `testone.ovdb` which is an OVIS configuration file that can be modified to test collecting and displaying data with OVIS. For example purposes the display shows 3 racks, 2 of which have two nodes each. Only one of the nodes will be used in this example and will collect data from your local machine. The physical display is shown in 29 (top) with the real component upon which data will be displayed popped out in the figure.

To use the localhost demo example:

1. Review the samplers to ensure that the metrics are correct for your system
2. Edit the addresses section of `testone.ovdb` where it is indicated to replace the data in the lines with that corresponding to your local IP address and MAC address
3. `mysql> create database OVIS_Testone`
4. `cd /path/to/ovisBuildDir`



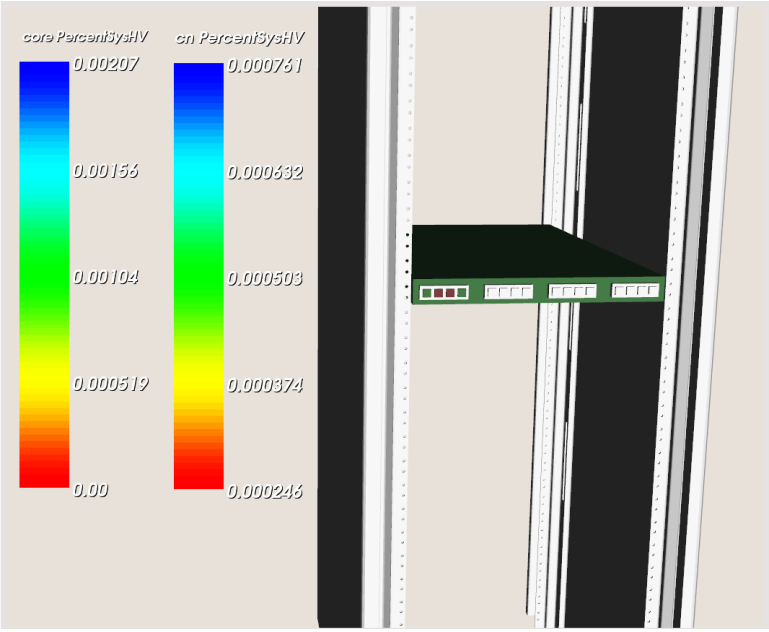




Figure 29: Node Level metric display (Local Sampler) from the testone.ovdb file (top); Core Level metric display (Remote Sampler) from the testonecpu.ovdb file (bottom).

5. `./bin/ovis-db -d -t 16383 -u mysql://ovis@localhost/OVIS_Testone -x /path/to/ovisSrcDir/data/testone.ovdb`

   (this will set up the correct tables in your database)

6. In a different window but the same directory run

   ```
   ./bin/shepherd --name=Testone \
     --database=mysql://ovis@localhost/OVIS_Testone
   ```

7. In a different window but the same directory run

   ```
   ./bin/sheep --name=Testone
   ```

8. In a different window but the same directory run

   ```
   export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/path/to/ovisBuildDir/lib
   ./bin/baron
   ```

### 7.2.2 Core level display

This case has 1 rack with one node which contains 4 CPUs each of which has 4 cores as shown in 29 (bottom). This illustrates both containment of components and the use of remote samplers. You can use this file in a manner similar to the previous example, but it will only be meaningful if you have a multicore machine.

This case illustrates use of the `ovMetricLinuxProcStatUtilRemoteSampler` which runs on the node and collecting data either to be displayed on the node (e.g., the overall utilization) or on the cores (e.g., the per core utilization). This is in contrast to the previous example where the `ovMetricLinuxProcStatUtilSampler` was used to collect the same information to be drawn to the node level only. Note that in the Node Level display example the per core metrics must all have unique names per core (e.g., CPU0UserPercUtil, CPU1UserPercUtil), where as on the Core Level display, they do not, as they are uniquely identified by the core. Note also that the Remote case requires the use of the `metric node map` as described in §4.3. More information on the specification of local and remote samplers for these examples is given is §4.3.1.

# 8 Additional Notes and Future Work

This section presents some additional notes and some planned enhancements.

## 8.1 Miscellany

Currently under development are enhancements to OVIS for handling job information including the display of jobs and idle times on the nodes and analyses invoked upon job ids. This feature will be incorporated in a future release.

Time series analyses will be included in a future release.

The Repeat Analysis capability (see §6.7) will be included in a future release.

Graphing/plotting capabilities will be included in a future release.

The Bayesian Modeling Analysis, referred to at ovis.ca.sandia.gov is not part of the release, and a patent application has been applied for on this part of the OVIS work.

OVIS is released open source under BSD license, which allows for the development of platform specific samplers or enhancements to OVIS to be kept for private usage. Please contact ovis-help@sandia.gov for more information.

## 8.2 Multiple Shepherds

The initial release of OVIS 2.0 supports a single database back-end.

Database insertion and analysis on a fully replicated database back-end (e.g. MySQLCluster) may work, but we have been unsatisfied with the performance of such databases in this application and consider this option to be an unsupported feature. In this configuration, multiple shepherds can be started and sheep attach themselves to a random shepherd from those that advertise. Each sheep then inserts data into its selected shepherd's database and the clustered database performs cluster-wide replication of the data upon insertion. Each instance of a shepherd inserts an entry in the **HaruspexIds** table (see §4.2.1) upon startup (and removes it when shutting down). When a haruspex request is made, the **HaruspexIds** table is used by each shepherd to determine its rank and the total number of participants; since the table is replicated, it contains a list of all available shepherds which can participate in analysis. Each shepherd then performs its calculation on a subset of the (replicated) metric data available to it. The metric data is partitioned by assigning equal fractions of components in the request to each shepherd.

We have begun development to enable distributed (non-replicated) database back-ends. Unlike the clustered database scenario, when sheep insert their values into any one of several shepherds no replication is performed. The **HaruspexIds** table is ignored in this case since shepherds perform

the analysis using all metric data available to them, as opposed to a subset computed from a rank and number of participants. The haruspex request and result data is replicated by OVIS using database triggers to detect insertions and updates followed by socket serialization of the rows in question. You can see our progress by preparing a build with the DISJOINT_DATABASE_BACKEND set to ON. This is for demonstration only; we do not support it. Only the descriptive and correlative haruspices have their request and result data replicated at this time. It will require an additional exception in your firewall for traffic to replicate rows.

Distributed databases will be supported in a future release.

# References


[1] J. Brandt, B. Debusschere, A. Gentile, J. Mayo, P. Pébay, D. Thompson, and M. Wong. OVIS 2: A robust distributed architecture for scalable RAS. In *Proc. 22nd IEEE International Parallel & Distributed Processing Symposium (4th Workshop on System Management Techniques, Processes, and Services)*, Miami, FL, April 2008.

[2] J. Brandt, B. Debusschere, A. Gentile, J. Mayo, P. Pébay, D. Thompson, and M. Wong. Using probabilistic characterization to reduce runtime faults on hpc systems. In *Workshop on Resiliency in High-Performance Computing*, Lyon, France, May 2008.

[3] J. Brandt, A. Gentile, J. Mayo, P. Pébay, D. Roe, D. Thompson, and M. Wong. Resource monitoring and management with ovis to enable hpc in cloud computing. In *Proc. 23rd IEEE International Parallel & Distributed Processing Symposium (5th Workshop on System Management Techniques, Processes, and Services)*, Rome, Italy, May 2009.

[4] Kitware, Inc. Visualization Tool Kit (VTK). www.vtk.org.

[5] Sandia National Laboratories. OVIS. ovis.ca.sandia.gov.

[6] TERASCALA. Terascala performance notes. www.terascala.com/pdf/Terascala Performance Notes.pdf. last accessed 2009-04-13.